2007

Mutually Adaptive Distributed Heterogeneous Agents for Data Classification

Joshua Alan Harlow

Mutually Adaptive Distributed Heterogeneous Agents for Data Classification

Master's Thesis Report

Joshua Alan Harlow
<jah8632@cs.rit.edu>
Department of Computer Science
Rochester Institute of Technology

Dr. Leon Reznik, *Chair*
Professor, Department of Computer Science

Dr. Roger Gaborski, *Reader*
Professor, Department of Computer Science

Dr. James Heliotis, *Observer*
Professor, Department of Computer Science

ROCHESTER INSTITUTE OF TECHNOLOGY

The Undersigned Faculty Committee

Approves the Thesis

Mutually Adaptive

Distributed Heterogeneous Agents

for Data Classification

Dr. Leon Reznik, Chairman
Department of Computer Science

Dr. Roger Gaborski, Reader
Department of Computer Science

Dr. James Heliotis, Observer
Department of Computer Science

Approval Date

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: Mutually adaptive distributed heterogeneous agents for data classification

Name of author: Joshua Alan Harlow
Degree: Master of Science
Program: Computer Science
College: GCCIS

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:
I, Joshua Harlow, hereby grant permission to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: Joshua Harlow Date: _____

Print Reproduction Permission Denied:
I, _____, hereby deny permission to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive
I, Joshua Harlow, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity. I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: Joshua Harlow Date: _____

Abstract

Efficient and effective data classification is one of the most fundamental problems of computer science due to a huge number of important applications. As such, there have been many models that are capable of doing some form of data classification using a large number of varied algorithms and architectures. However, few share properties that implicitly take advantage of the distributed environment where classification has its most potential.

Therefore, to resolve this dilemma in this thesis a generic classification architecture composed of mutually adaptive distributed heterogeneous agents was designed, implemented and experimentally analyzed. The system defined creates a multi-agent distributed architecture whereby individual agents cooperate and collaborate with each other to autonomously perform classification on arbitrary data using a hierarchical paradigm.

The main contributions of this work are the following: agents follow hierarchical processes to form an organization which can autonomously train, test, assign and evaluate work of other agents. Through experimental analysis it was found that this design improves the classification accuracy relative to previously implemented classification algorithms and architectures. Furthermore this analysis shows that the generated architecture will autonomously adapt to previously unlearned data and responds to failures with little or no degradation of classification quality.

Contents

This page intentionally left blank.

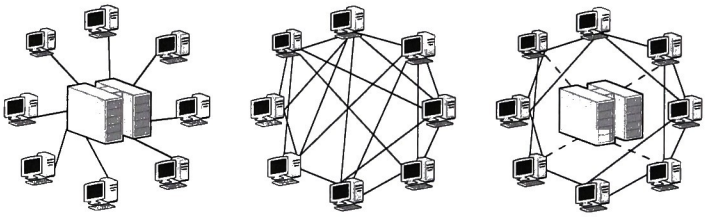
Chapter 1

Introduction

Intelligent systems, containing "intelligent entities that are capable of independent action in dynamic, unpredictable environments[26]", have recently become an increasingly developed and fast growing segment of the artificial intelligence community. These systems have been developed for use in a diverse area of applications, varying from online web data mining[41] to security related areas such as intrusion detection[1, 50]. Each system shares similar concepts in that some intelligence mechanism is taken advantage of to allow the components to achieve some predetermined goal. Furthermore as distributed systems have become increasingly popular, these intelligent systems are often also distributed, whether through classical distributed models or through a progressively popular model of distribution through separated agents. Additionally since input and outputs are usually present in such systems, whether completely autonomous or partially user directed, in many cases principles from the concept of knowledge discovery in databases can be applied, whereby "nontrivial extraction of implicit previously unknown, and potentially useful information (is extracted) from data[18]." Therefore, as these concepts form the backbone of this thesis they will be introduced in the following section, allowing the heart of this thesis, whereby intelligent entities are organized under a hierarchical paradigm, to operate.

1.1 Distributed systems

The computational environment has recently been shifting into a new era, one where "distributed systems are everywhere[7]" and shared resources or services have become increasingly important. These distributed systems bring many advantages including reducing the impact of individual component failures, increasing reliability and taking advantage of *social* activities to aid in whatever processes are being performed. With distributed architecture models such as the centralized client/server (see figure 1.1(a)) model, the hybrid peer-to-peer model (see figure 1.1(c)) and the pure peer-to-peer model (see figure 1.1(b)) the classification of what is distributed has also become increasingly ambiguous. One common attribute that is shared is the concept of social connections, whereby either through a central server or through peer links some communicative activity is performed to obtain some desired goal, whether this be a simple send and acknowledgment or a more complex n-directional protocol. As can been seen from the different models, this communicative activity is handled differently by each variation on the distributed environment.



(a) Client-server (b) Pure peer-to-peer (c) Hybrid peer-to-peer

Figure 1.1: Distributed system models

The simplest model, that is most common with distributed systems is the centralized client/server system shown in figure 1.1(a). This model manages the communicative activity by having clients interact with servers in order to access shared resources or message other peers. Notable this model has inherent disadvantages, including the centralization of all resources at a individual server, which reduces the ability of this type of system to

deal with failures gracefully. Another fault with this model is the responsiveness of the individual servers, i.e. if to many clients connect to a single server a denial of service may result, whereby clients do not receive the desired resources in a timely manner (if at all). On the other hand this model is relatively simple and easy to design in that a client must only be aware of the server that its resources will be made available from.

The second type referred to as the pure peer-to-peer model, see figure 1.1(b), is a distributed model that lacks the centralized server of the client-server model. These types of model instead offer a completely scalable model where each peer is aware of a certain number of other peers who are aware of a number of other peers and so. In this model each peer can act as a client or server as needed thereby not necessitating a central server. This model has gained enormous popularity recently through the exposure of networks such as bittorrent* and distributed hash table implementations with corresponding routing overlay networks such as Pastry[52] and CAN[51]. Unfortunately this model although offering inherently limitless scalability and potentially unlimited resources has problems with organization, coherence, search and security when tens, hundreds or thousands of peers are accessing resources provided by other peers.

To correct these problems the third type of network was created as a combination of the client-server model and the pure peer-to-peer model and is shown in figure 1.1(c) whereby instead of being completely decentralized this hybrid model offers a central peer which plays the role of an index (similar to a book of yellow pages). This index peer provides a centralized resource where peers can search for and discover other peers without requiring a direct channel to the desired peer. The index peer accomplishes this by storing information about each peer (such as what services they can provide) but does not store any actual resources therefore maintaining the distributed nature of a pure peer-to-peer network but generating less traffic and more security since authentication and registration can occur as needed to gain access to the index peer.

*http://www.bittorrent.com/

These models are especially relevant to this thesis since the agents in this system are inherently distributed and require the ability to easily search and locate other agents services and capabilities. Since the location of services and capabilities is extremely important in an agent system, J.A.D.E. the agent framework that this thesis will be using, and which will be described in chapter 3 will apply a network using the hybrid peer-to-peer model shown in figure 1.1(c). This distributed model ensures that each agent can search and locate each other, simply by ensuring they register their services, languages and other aspects of their capabilities with the central index peer as described previously.

1.2 Agent systems

To understand how an agent is going to be used in this thesis we must first understand what typically defines an agent. In a popular multi-agent text an agent is a "computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives" [63]. This extremely loose definition describes the abstract view of an agent, as can be seen from figure 1.2. From this figure we can note that this agent will take some input from whatever environment it is situated in and applies some sort of processing to that input, therefore giving it partial control over its environment. Thereafter it will potentially modify the environment by performing an action to react to its sensory inputs.

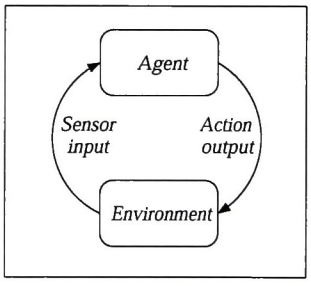


Figure 1.2: An agent in its environment[63]

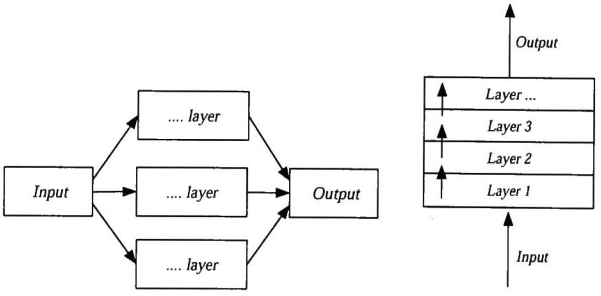
From this simple definition we can therefore attempt to define what an intelligent agent is. Unfortunately the description of what exactly describes intelligence is hotly debated in the A.I. community. As stated in [64, 63] most likely an intelligent agent has characteristics that make it autonomous, social, reactive and pro-active. Being autonomous would involve the same concept as described in the "basic agent" whereby an agent need not interact with humans or other agents in order to accomplish its goals. Secondly, having social characteristics would involve the ability to interact with humans or other agents in an agreed upon language. Thirdly an intelligent agent must be reactive in that they should "perceive their environment and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.[63]" Finally and most importantly an intelligent agent must be pro-active in that it must "exhibit goal-directed behaviour by taking the initiative in order to satisfy (its) design objectives.[63]"

Now that the definition of an intelligent agent has been solidified, the next step in understanding agents is determining the architecture the agent shall be using for its internal systems. There traditionally exists four base architectures[63], these being logic based, reactive based, belief-desire-intention based and layered architectures. The logic based architecture follows the traditional approach whereby an agent takes some *symbolic* description of its environment and using its current behaviour and first-order predicate logic[53] modifies its environment therefore changing its *symbolic* description. This type of architecture has many advantages encompassing such benefits as being a well theorized, logically clean, well tested and one of the oldest known applications of artificial intelligence. Unfortunately though there also exist many problems which often deter this architecture from being used in an agent platform, such as having difficulty representing temporal information and being to computationally complex for environments which are dynamic.

To overcome these limitations the second architecture was proposed, whereby instead of the complex architecture of the logic based system a reactive architecture is proposed (made popular by Rodney Brooks subsumption architecture[5]). This architecture simply

implements behaviours as reactions to situations therefore attempting to allow intelligent behaviour to *emerge* from simpler behaviours. This architecture gains many advantages over the logic based architecture such as being simplistic, computational tractability and having robustness against failure. Unfortunately though this architecture has its share of problems as well including being unable to learn from past experiences and being limited inherently to a short term view of the surrounding environment.

The third type of architecture takes a completely different view on how an agent should process and manipulate its environment. This architecture is founded around two important processes, "deciding what goals we want to achieve (deliberation) and how we are going to achieve these goals (means-ends reasoning).[63]" Initially these goals are formed as persisting intentions, which then direct the agents actions and future reasoning. Then once these goals are established some type of action is selected in order to achieve these goals, which may involve more intentions being created. As a rather intuitive model this model abstracts the daily processes we all make in selecting intentions, goals and actions to achieve these intentions. Unfortunately, as with all models there are downsides; one downside of this model is the complexity of implementing these intentions, goals, and actions correctly.



(a) Horizontal model (b) Vertical model

Figure 1.3: Layered architectures

The fourth and final model deals with a combination of reactive and pro-active layers

whereby a control flow hierarchy of interacting layers is created, in either a horizontal or vertical pattern. Horizontally layered models (see figure 1.3(a)) directly connect the software layers with the input and the action output and produce potentially actions which may have outcomes internally to the agent or externally as messages. Vertically layered models (see figure 1.3(b)) on the other hand have sensor input and action output handled through passing of information vertically as needed to reach the action output layer. These models although allowing for reactive and pro-active behaviours to exist concurrently and having "conceptual simplicity" have problems with competing behaviours. Solving this requires an application of some type of mediator to decide which behaviour should be invoked at any given time, therefore increasing design complexity as any interactions between layers must be considered and handled by the mediator or mediators.

The importance of intelligent agents to this thesis is inherently obvious, but the selection of architectures is not as inherent. Throughout chapter 5, certain design decisions will be made that select reactive or layered architectures as the foundation of different agents in order to obtain the minimal amount of complexity during implementation.

1.3 Knowledge discovery in databases

The knowledge discovery in databases (K.D.D.) process, as can be seen from figure 1.4 plays an important role in the analysis and processing of data as "we are drowning in data, but starving for knowledge" [28]. The processes that compose K.D.D. each have major effects on the quality of the data outputted, with selection, preprocessing and transformation (data cleaning/conversion) processes comprising the most work (60% of the effort) and having the largest impact on the entire process. These processes described here briefly from beginning to end, starts with selecting a large enough sample in the selection process to preprocessing where any "repairing" of the data-set occurs to the process of transformation where the data is made into a suitable format for the data mining engine (such as string to numerical transformations) then to take over. Thereafter data-mining algorithms will

attempt to perform some type of analysis on the data, potentially performing classification, clustering, pattern identification or some other type of data analysis as desired by the end users of the system. Once this process has completed in most cases a large amount of new data will be generated, hopefully less than the original input, which then must be analyzed to gain some type of useful knowledge (or the process may repeat with new conditions).

The connection with this thesis is partially in the first three processes, which must be performed before the theses architecture can be activated. Thereafter once these three processes are completed this theses agent system will then use that data to operate its agents on. Subsequently the system shall provide the data-mining aspects (through classification) and attempt to aid in the interpretation and evaluation processes of K.D.D. by displaying progress and outputs in a user friendly format (see chapter 5 and appendix B).

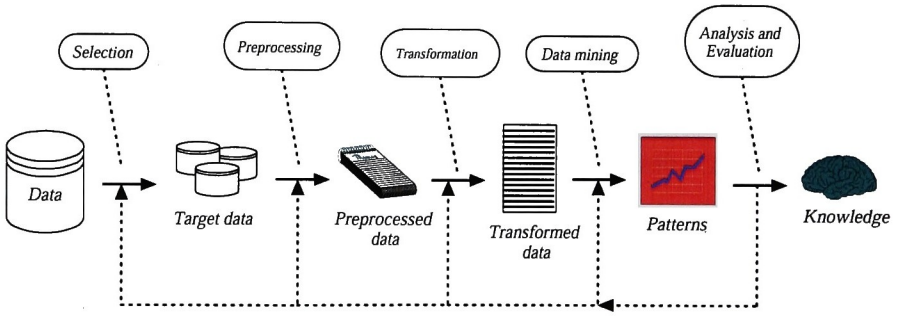


Figure 1.4: K.D.D. process

Chapter 2

Problem description & research objective

2.1 Problem description

Efficient and effective data classification is one of the most fundamental problems of computer science due to a huge number of important applications. Over years many algorithms both statistically oriented and based on heuristics have been developed, including but not limited to k-means, neural networks, and decision tree algorithms such as CART, C4.5, ID3, ID4, and ID5R (see chapter 3). Typically these algorithms are applied in environments which are not distributed and use statically defined data-sets[19, 56, 49, 3]; environments which are unrealistic in the occurring distributed revolution.

This revolution brings with it situations where the data involved in classifications is dynamic, the resources distributed with a low resistance to failures, thereby making it very difficult to apply conventional classification methods. Previously these situations have been approached by using a fixed set of classification algorithms which react to the surrounding environment, with adaptation occurring at an algorithmic level. Unfortunately, the main design problem with this approach is that since these algorithms are isolated, they therefore have not been taking full advantage of their capabilities as a group, allowing those

algorithms to work together to perform classifications. Concurrently the necessity for a framework which can accommodate and adapt to these situations has become increasingly relevant, with an even greater level of relevancy to areas such as intrusion detection which depend on distributed situations. In order to accommodate these capabilities, the algorithms can be distributed in some manner and managed as a group in some form; which if done intelligently has the potential to improve classification accuracy and adaptability.

Previous multi-agent systems have been proposed which examine the mutual training process, and have shown the process to be beneficial (see chapter 3), but there does not currently exist any such system which can use a mutually training process to aid in generic classification. Therefore to overcome the previously limiting situations, this thesis proposes a solution by applying a mutually training process which has a structural design that mimics the neocortex region of the mammalian brain, which is thought to provide much of the intelligence mammalian species are endowed with[29]. This neocortex is the part

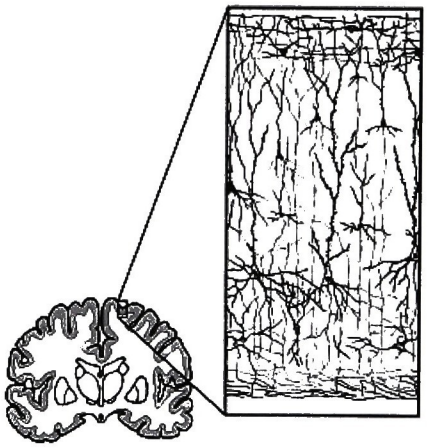


Figure 2.1: Neocortex and cortical columns[8]

of the brain we are most familiar with, as it is the "wrinkly" exterior that is most often represented in pictures (see figure 2.1). It is composed of six layers, which has over 100 billion cells, each with 1,000 to 100,000 synapses, all packed into a set of layers that has the

thickness of a paper napkin[40]. These layers are thought to perform multiple functions, with different layers and interactions between layers aiding in sensory perception, spatial reasoning, vision, touch, balance, movement and emotional responses[40]. These functions are enabled through an application of neurons structured into cortical columns[25, 42], forming a hierarchal structure between layers, which is distributed throughout each of the different regions of the neocortex. As this structure has been proven extremely flexible and adaptable, this thesis theorizes that one way of gaining these capabilities is to follow a similar distributed hierarchy. By following this design, classification systems should gain greater adaptability, tolerance to failures, and provide for flexibility by incorporating algorithms into disjoint but connected intelligent regions.

2.2 Research objective

The objective of this thesis is to develop algorithms and an implementation of a mutually training distributed agents, employing research, previous work and innovations as necessary to accommodate progress. The goal is to develop the methodology for a design and implementation of a framework which will apply multiple cooperative agents working together to efficiently classify data, with potential applications to a variety of different problem areas.

In order to achieve the objective the following plan will occur:

- Develop cooperative hierarchy principles and algorithms describing the systems functionality
- Implement the system, using the principles and algorithms, as a software framework suitable for easy adaption to different application areas
- Test the design on intrusion detection and waveform identification data-sets
- Compare the system designed and accompanying experimental results to conventional algorithms and methods wherever applicable

The algorithms will be centralized around forming a previously untried hierarchy, which mimics the neocortexes regions and its cortical columns. This hierarchy will be composed of multiple agents, each with duties that mimic the way the various regions of the neocortex work together to allow for emergence of intelligent behaviour. In order to achieve this objective it was decided that a design would be selected which follows an organization's structure and is composed of distributed agents who perform work related tasks, with the tasks forming a hierarchical dependency on other tasks or agents; thereby constituting a cooperative hierarchy.

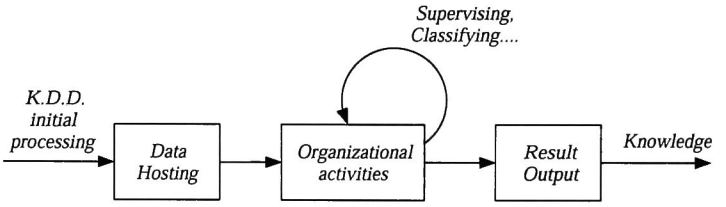


Figure 2.2: Framework overview

The algorithms required to perform these activities, once completed, will then be incorporated into a larger classification framework involving data hosting, the organizational processes and result output (as can be seen in figure 2.2). Thereafter a set of agents will be constructed which will perform the actions necessary to invoke these algorithms, in a distributed manner, thereby increasing the fault tolerance of the system as a whole, as an individual agents failure will cause a graceful degradation in the quality of classifications (similar to the neocortexes distributed redundancy). Although the goal of this thesis is to develop these algorithms/agents and an implementation, additionally user friendly graphical applications will be incorporated. These will allow for anyone without an in-depth knowledge of the agents and underlying algorithms to be able to use the developed agents and applications (see appendix A).

The overall hope is that if this process is done intelligently, with a organizational structure that follows a hierarchal process similar to the neocortex, the classification process

will proceed with increased accuracy, adaptability and a greater tolerance to failures. This solution, if successful, should therefore have an immediate impact on areas which are already dynamic, distributed, or have a high failure potential, such as intrusion detection and waveform identification. Concurrently the developed design, framework, algorithms, and implementation could also open up new areas and applications where classifications can be performed.

2.2.1 Outline

In order to accommodate these objectives the following outline for this thesis report will be conducted. First off there will be a review of relevant literature and related topics in the area in which this thesis will innovate in chapter 3. This will overview a subset of previous mutually training agent development and classification algorithms and discuss if and how they are relevant to this thesis. Following this, in chapter 4, will be an overview of the technologies used to aid in the implementation of this thesis, including various programming resources and libraries. Directly after this will be the design and functionality details which will be provided in chapter 5. This chapter will go into the high level design, and lower level implementation details of the generated agents and algorithms generated to accomplish the defined research goals. Thereafter design of experiments, experimental results and comparisons shall be provided in chapter 6. During this chapter, the experiments to be performed will be described in detail along with the results achieved by running those experiments, therefore allowing for an analysis of this theses methods. Subsequently, following this experimental analysis will be potential future work being provided in chapter 7. Finally to end thesis conclusions & implications will be discussed in chapter 8. For those interested in using the developed applications the accompanying user guide and G.U.I. screenshots in appendices A and B can be referenced. These appendices provide a guide and screenshots to learn how to use the agents and what needs to be done to get them up and running.

This page intentionally left blank.

Chapter 3

Background

As we "stand on the shoulders of giants" (Isaac Newton) most new technologies and in general any innovative idea, has some underlying core of previous work. These pieces of work may be a source of resources or a source of contrast on which new ideas or variations are built upon. This thesis is no different, with sources of work which shall follow in this chapter that have inspired this work, either by having flaws which this thesis attempts to solve or remedy to a certain degree or by having a beneficial idea which this thesis will apply or be inspired from.

3.1 Agents

One of the first mutually learning agent implementations [62] proposed two learning scheme labeled A.C.E. (action estimation) and A.G.E. (action group estimation) whereby mutually learning occurs in reactive agents. The first algorithm, A.C.E., proposes a system whereby three activities occur, determination, competition and credit assignment. The first activity determines a set of actions which the agent believes it can carry out relative to its environmental state and current knowledge. Then the second activity occurs whereby the agents "compete for the right to become active" by bidding on the "selection of actions that are

actually carried out." Thereafter the action with the highest bid is carried out and a credit assigning process occurs[62] to weight the selected actions against potential future actions. The second algorithm, A.G.E., is similar to A.C.E. but modifies the competition phase by competing for groups of actions instead of solidarity actions. Thereafter the same bidding and credit assigning processes occur, but instead of a single action being carried out the whole group of actions are performed. Following this description the authors compare their results against a random search algorithm using both algorithms to move blocks around in the traditional blocks world problem (choosing actions sets either randomly or through the proposed algorithms). Their results showed that the learning schemes perform significantly better over random walk schemes and were able to learn stable sequences of action sets. This therefore positively enforces the ideology that agents working together, even when those agents are reactive, can perform better than agents choosing random actions sets.

In [57], the author Ming Tan investigates "given the same number of reinforcement learning agents, will cooperative agents outperform independent agents who do not communicate during learning?" Using reinforcement learning[33] the author attempts to determine the performance of n independent agents with n cooperative agents and identify any trade-offs which may result. Therefore as reinforcement learning typically applies *q-learning*[61] the author performs analysis to compare the strategic performance of agents acting in a hunter-prey simulation. In this simulation each agent is setup to be a hunter or prey, with a limited depth of field, and random start location. To implement a reinforcement policy when the hunter group captures a prey they receive a $+1$ reward, and when they move but do not capture a prey they receive a -0.1 reward. Through multiple case studies and experimental tests the author empirically determines that cooperative agents can learn faster and converge sooner than independent agents. The paper also shows that the communication needed for this cooperation comes at a communication cost, providing tradeoffs which this thesis must also analyze.

In [24, 23] the authors present a cooperative multi-agent learning algorithm whereby

the agents first go through a training session and "are then able to choose their actions by generalizing what they have learned." Using a teacher-learner model, the authors propose a trainer/teacher agent who instructs "students" on how to behave. In this model each agent can be a trainer or a learning agent, where the trainer tries to train the learner for its own benefit. The trainer also grades the learning agents behaviours, through a feedback mechanism which therefore affects the learning agents next action. The learner agent subsequently computes its satisfaction level based the current world state, the agents goals and its feedback to determine which action to choose next to increase its satisfaction level. In the article the authors apply their algorithm, comparing agents who choose their actions randomly versus cooperative agents using a traffic light simulation. Through experimental results the authors were able to give show results showing that cooperative agents were able to achieve 0% collisions and cooperated in 49% of the cases, with the downside that a large queue of cars was maintained.

In [45, 44] the authors, Luìs Nunes and Eugènio Oliveira attempt to determine "how can agents benefit from exchanging information during learning?" Interesting in this set of papers is that the authors consider the possibility that the agents may have different structures and different learning algorithms, therefore not being a homogenous set of agents. In order to attempt to determine the answer to this important question, the authors generate agents whereby different learning algorithms are used (random walk, evolutionary algorithms, simulated annealing and *q-learning*). Similarly as [24, 23] the authors use a traffic light simulation, whereby the agents "observe the state of (their) environment for their local scenario and decide on the percentage of green-time to attribute to the North and South lanes.[44]" During this process the agents exchange advice by broadcasting their best average result for each epoch whereby then at the beginning of the next epoch agent i will compare all results to determine which agent has the highest average result to then seek advise from. Thereafter the "advisee" sends its state to the "advisor" who then runs that state and returns a result using its learning algorithm, therefore giving supervised

feedback to the "advisee." After running many experimental trials with the various learning algorithms the authors determined that "advice-exchange seems to be a promising way in which agents can profit from mutual interaction during the learning process" but they also found that "bad advice, or blind reliance, can hinder the learning process, sometimes beyond recovery."

In [32], the authors analyze the effect of communication, when each agent has "access to a small set of local information and through experience learns to communicate only the additional information that is important." In order to analyze the effect of communication the authors develop a predator-prey simulation, whereby the goal is to have four predator agents surround a prey on all four sides in a grid-world. To allow for communication to be analyzed the authors develop a blackboard[9] like architecture whereby all agents post binary messages (messages only containing $\{0, 1\}$) to a message board. Therefore using this message board the authors develop the predator agents by applying genetic algorithms[22] to evolve different agents; whereby each evolution produces different languages and strategies. Applying these agents through various life-cycles, the authors determined that "genetic algorithm(s) can evolve communicating predators that outperform the best evolved non-communicating predators, and that increasing the language size improves performance."

These previous seven papers provide a conclusive answer to the question of whether cooperative learning is beneficial and even achievable, as they show that social agents cooperating and communicating together can exist and do perform better than independent agents. This result is imperative for this thesis since the agents in the implementation of this thesis will be inherently communicating and cooperating to accommodate classification progress. Also these papers bring forward areas which this thesis must be aware as the effect of blind reliance and the cost of communication are areas which could hinder this theses classification capabilities.

3.2 Classification

As far as classification algorithms are concerned, there have been a large range of different algorithms over the past 20 years which have attempted to define methods for classification. These different methods span a large range of artificial intelligence such as decision trees, rule induction, neural networks and clustering. As the goal of this thesis is to provide a dynamic classification framework using agents, a selection of these previous classification works and associated methods will provide a sense of contrast/comparison as most do not provide easily distributable, adapting algorithms for data classification. As for the algorithms which do provide an adapting view of data classification, a subset of these will be applied in the design stage (see chapter 5). Additionally in chapter 6 certain experiments will be performed allowing for a comparison, if possible, of the capabilities of this thesis with previously implemented classification algorithms and applications.

3.2.1 Decision trees

One of the approaches for generating classifications has been to generate decision trees for the instances being classified. These decision trees are "tree" like structures whereby each node in the tree represents either a specific class or some type of test that splits the tree into separate subtrees with leaf nodes always assigning a classification. These decision trees are frequently used for classification as they provide a "divide and conquer strategy for object classification."[58]

One of the well known algorithms for building these trees is the ID3 algorithm which attempts to use information entropy defined in [55] and information gain to determine the "importance" of the data at each level of the tree. On a basic level the entropy calculation, shown in equation 3.1, determines the information gained by a certain attribute by being provided a collection S and the proportion of class I in S. Similarly the information gain shown in equation 3.2 attempts to determine how well a attribute of a data-set splits

the data-set into different classes. This occurs through the application of the entropy formula, but also taking into account S_v which represents the subset of the total data-set for which attribute A which has value v therefore determining the gain that this attribute A offers. The ID3 algorithm proposed takes a set of training instances and recursively selects attributes of those training instances, calculating their entropy and gain and selects the attribute with the most gain as the current tree node. This process repeats with the attributes not partitioned until no more features are available to partition or the algorithm perfectly classifies the training instances. As can be inherently seen this algorithm is non-iterative, in that as new data is obtained the whole tree must be rebuilt from scratch.

$$Entropy(S) = -\sum p(I)\log_2(p(I)) \tag{3.1}$$

$$Gain(S, A) = Entropy(S) - \sum \left(\left(\frac{|S_v|}{|S|} \right) Entropy\left(S_v\right) \right) \tag{3.2}$$

Another popular decision tree algorithm, C4.5[49] builds upon the ID3 classification algorithm by providing a number of extensions. These improvements include allowing unknown or missing attribute values, tree pruning to reduce the tree size, attempting to avoid over-fitting, handling continuous values (i.e. time) and generally improving computational efficiency. This algorithm although providing many advantages for software has limitations in that its pruning methods are stated by the creator (Ross Quinlan) as being a "heuristic with questionable underpinnings." The C4.5 also by being a relative of ID3 does not allow for iterative tree building, therefore also limiting its usage to areas where static data is being classified or where the time to rebuild the entire tree is inexpensive or need not be a concern.

Another non-iterative decision tree algorithm which is used quite often for constructing decision trees is CART[4]. This algorithm uses a different model than ID3, whereby instead of information gain a probabilistic model is used instead. The CART algorithm has various strengths[65] over C4.5 and ID3 such as it can deal with data with high dimen-

sionality, handle missing values and is not be affected by outliers. Unfortunately it also has weaknesses[65] as it is not an iterative algorithm and the variables in CART are not required to follow any statistical distribution.

In order to allow for iterative tree building, quite a few algorithms building off of ID3 have been proposed. One of the first attempts was the ID4 algorithm proposed in [54] whereby the algorithm maintains a score at each possible test attribute. This score is updated as new instances arrive and results in a shifting of test nodes as lower scoring nodes get replaced with higher scoring nodes. Unfortunately this algorithm also discards the subtree of lower scoring nodes whenever they are replaced therefore causing chaotic movement and lose of subtrees (and the information they contain) as the tree attempts to stabilize. These problems prevented usage of this algorithm as some concepts are unlearnable if the tree never reaches a stable state [58] .

A solution to ID4's problems was proposed in [58] whereby the authors provide another attempt labeled ID5R. This article provides a well written overview of the ID4 algorithm and goes into depth theoretically and empirically on how their algorithm modifies the basic methods of ID4 to provide for a practical restructuring procedure. This is accomplished by the following method; instead of discarding subtrees this algorithm performs a "pull-up, a tree manipulation that preserves consistency with the observed instances.[58]" Overall this algorithm, proved through experiments, was found to be less expensive for updates than applying the ID3 algorithm repeatedly and overcame the technical problems which were inherent with the ID4 algorithm.

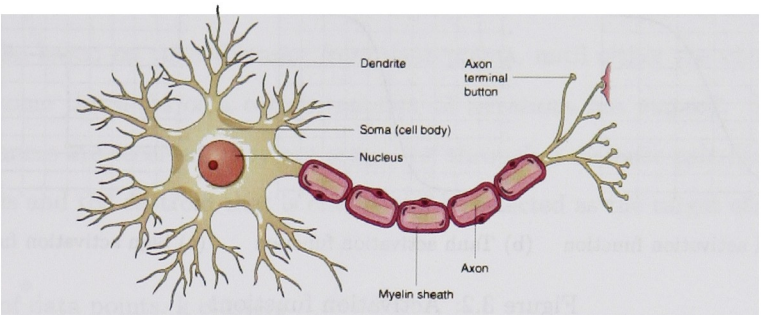
A third iterative algorithm proposed in [59] applies a slightly different methodology to iterative tree building whereby a mapping from a previous tree to a new tree is generated, using similar test information as ID4 and ID5R stores. This algorithm labeled I.T.I. for iterative tree induction performs the following over-viewed procedure[59]. As each instance arrives, it is passed down to the branches until a leaf node is reached, updating at each node passed through the test information and marking that node stale. After this new instance

has been incorporated visit the stales nodes and modify the test at that node to reflect the changes that instance may have caused. This algorithm and its slightly modified revisions presented in the article show through experimental results that compared to C4.5, accuracy is equal to or better. Unfortunately the cost of updating data-sets and corresponding instances with large amounts of numeric values was found to be extremely costly.

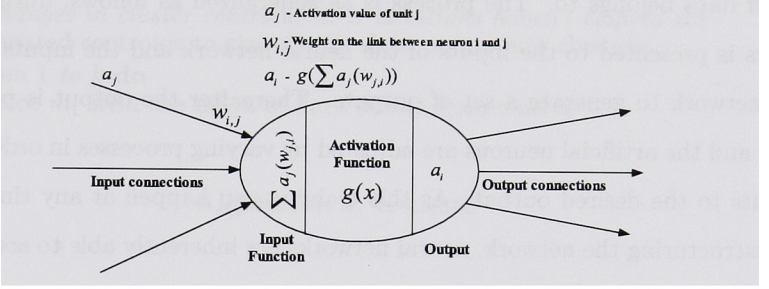
3.2.2 Neural networks

A second method for generating classifications is using the capabilities of computational neural networks to internalize classifications as a reflection of the neuron structure[30]; one of their many applications. These artificial neural networks act as approximations of the neuron structure that acts as the main component of our nervous system. As artificial neurons (see figure 3.1(b)) directly map to biological neurons (see figure 3.1(a)) both have a similar structure and perform many of the same responsibilities. These biological and computational responsibilities include being the main processing elements and transmitters of information in all vertebrate animals[31] and computational environments where they are applied. These processes occur through a complicated communication process using electrical and chemical interactions for biological neurons or for artificial neurons a complicated structure and various algorithm are used to simulate the biological neurons processes. As the topic of neural networks covers whole books an overview of the structure and the background of neural networks will be provided in the following section.

The biological process starts from one neuron's terminals whereby neuro-chemicals are released which then pass through the synaptic boundary and are potentially accepted by another neurons dendrites. Then through a complicated process involving the chemicals transmitted and the change in ionic balance inside the receiver these chemicals may activate the action potential of the receiving neuron, causing a potential retransmission of neuro-chemicals to another neuron through the receivers axon terminals. It is through this process that initially inspired the development of artificial simulations of the neuron and its



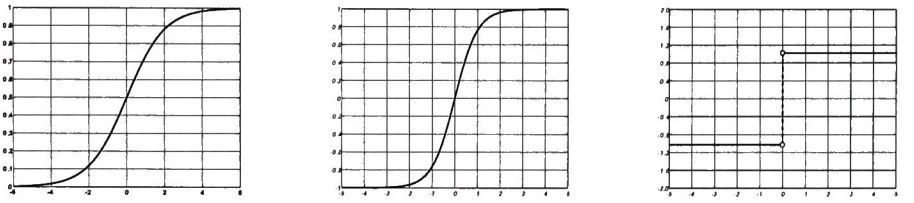
(a) Biological neuron structure[31]



(b) Artificial neuron structure

Figure 3.1: Neuron structures

complicated network of interactions. These artificial processes follow a similar but simpler procedure, whereby a number of artificial neurons are logical connected together through different types of synapses. The input synapses to a artificial neuron are then processed in some manner and placed through an activation function to produce an output which is then logically sent to a receiving artificial neuron. Depending on the different type of network model being used and the area and type of data being processed these activation functions can vary from a sigmoid function (see figure 3.2(a)) to a hyperbolic tangent function (see figure 3.2(b)) to a sign function (see figure 3.2(c)) to various other functions that modify the inputs in various manners. As biological neurons adapt to changes, artificial neuron also inherently adapt to changes in the input data. This learning process occurs through supervision and reinforcement of a set of training data which contains the classes that



(a) Sigmoid activation function (b) Tanh activation function (c) Sign activation function

Figure 3.2: Activation functions

each record of data belongs to. The process is as generalized as follows, during learning a set of inputs is presented to the inputs of the neural network and the inputs are passed through the network to generate a set of outputs. Thereafter the output is processed in some manner and the artificial neurons are adjusted by varying processes in order to better map the inputs to the desired output. As this training can happen at any time, without completely restructuring the network, neural networks are inherently able to accommodate domains which are dynamic and iterative.

3.2.3 Clustering

A third method for generating classifications is using unsupervised (no training is performed) algorithms in an attempt to predict classifications by using some type of "natural" grouping. These algorithms work by exploiting this "natural" grouping to separate instances into a varying numbers of p groups (either user specified or autonomously determined) with these groups therefore specifying the classification.

One of the most well known algorithm for performing clustering labeled k-means[37] allows for straightforward non-iterative classification procedure which will be applied in this thesis to determine the centriod of classification results received. This clustering algorithm works by a rather simple procedure shown in algorithm 1 and visually in figure 3.3; whereby a set of data points is given to the clustering algorithm and a value k for how

many clusters to generate. Thereafter the clustering algorithm will continue rearranging the cluster centroids, based on their distance from data points, until either the clusters have shifted below some threshold or a certain number of iterations has elapsed. Thereafter when new instances are received, they are compared through a distance calculation to all cluster centroids and the centroid that is closest will be selected as the target class of that instance.

Data: Set of data points, k clusters
Result: Set of cluster centroids
Make initial guesses for the cluster centroids m_1, m_2, ..., m_k;
while *no changes in cluster centroids or n iterations haven't elapsed* **do**
  Use estimated centroids to classify the data points into clusters;
  for *i from* 1 *to k* **do**
    Replace m_i with the mean of data points from cluster i;
  end
end

Algorithm 1: K-means

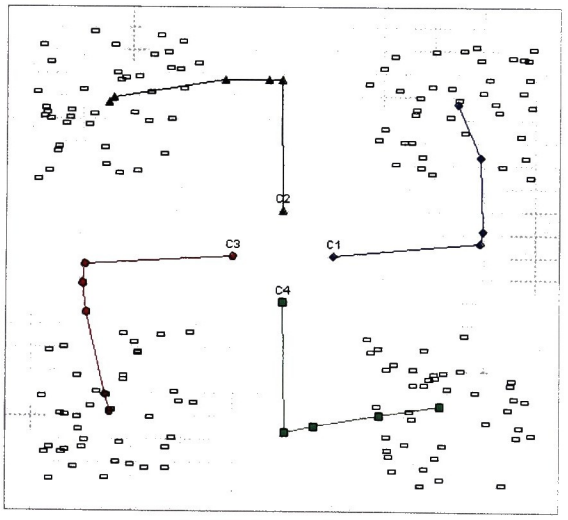


Figure 3.3: K-means visualization

This page intentionally left blank.

Chapter 4

Technologies

4.1 Java

The Java 1.5 set of technologies were chosen as the main programming language and run-time environment for the implementation of this theses code base. This language was chosen due to its garbage collection, cross platform capabilities and object oriented model which allows for straightforward creation of interfaces and abstract classes, therefore encouraging good design practices. An additional feature new to Java 1.5 and relevant to this thesis is the new concurrency classes and generics. These new resources allow for simplified concurrent programming, as this thesis is highly multi-threaded, and for type safe collections and other storage and parameter typed classes/functions, therefore allowing more software bugs to be caught at compile time. Also since Java is nearly 100% object oriented this allows me to use it as the basis for many software design patterns[20] that this theses implementation (see section 5.2) will take advantage of to create a modular, robust and expandable architecture. Another important factor was that J.A.D.E. (see section 4.3) and Joone (see section 4.4), libraries used by this thesis, are both implemented natively in Java.

4.2 XML

XML (see figure 4.1) or the Extensible Markup Language was chosen as the set of technologies for all configuration files relating to the agent platform and any data-sets involved with the platform. This markup language "defines a generic syntax used to mark up data with simple, human-readable tags"[39] and is designed to work across applications, platforms and environments. The markup permitted in the configuration files also can be easily structured in that an XML schema specifications can be provided which will limit what markup can be used. Therefore using this technology which has extensive support in Java allows for easily changing runtime properties for the agents themselves or for the data being used in those agents. This avoids the problem whereby constants or properties are fixed inside the agents, a bad design practice, making it difficult to change those configuration values when needed.

```
<?xml version="1.0"?>
<product barcode="2394287410">
 <manufacturer>Verbatim</manufacturer>
 <name>DataLife MF 2HD</name>
 <quantity>10</quantity>
 <size>3.5"</size>
 <color>black</color>
 <description>floppy disks</description>
</product>
```

Figure 4.1: XML Example[39]

4.3 J.A.D.E.

In order to design this theses agents a framework was chosen that provides the basic middleware and related Java classes relating to agent technologies, therefore avoiding having to recreate a piece of software already know to be quite stable. This toolkit named J.A.D.E.[10]*, standing for the Java Agent DEvelopment Framework, acts as the middle-

*http://jade.tilab.com/

ware (see figure 4.2) between agent code and the underlying structure needed to provide communication and services for those agents to take advantage of. The protocols for communication and related services used by J.A.D.E. directly correlate to the standards created by an IEEE standards organization labeled F.I.P.A. standing for "The Foundation for Intelligent Physical Agents." This organization formed in Switzerland during 1996 and recently accepted as an IEEE organization in 2005 "promotes agent-based technology and the interoperability of its standards with other technologies."[†]

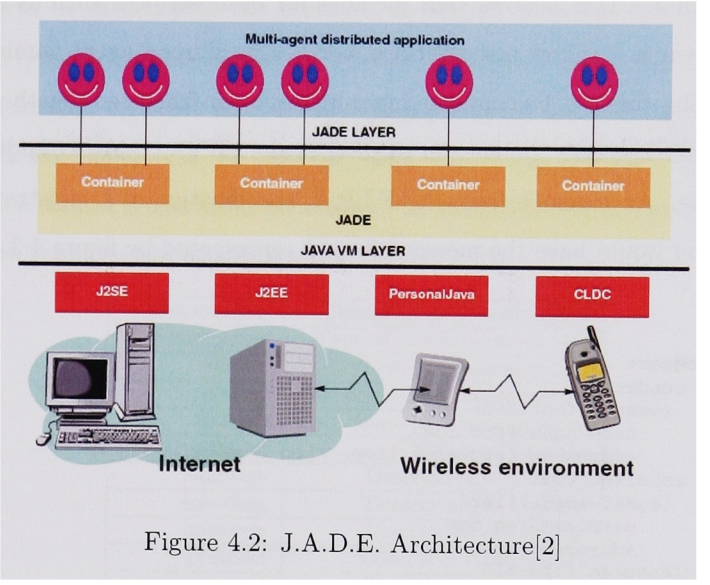


Figure 4.2: J.A.D.E. Architecture[2]

This framework provides a hybrid peer-to-peer model (see figure 1.1(c)) applying a concept whereby all agents join containers, this being an instance of the J.A.D.E. runtime. For these J.A.D.E. runtime containers to work, a single Main Container must initially exist which sub-containers will be put in contact with only on cache-misses, i.e. when agents are messaged that the local container does not know exists. These container instances abstract the complexity of communication and the diverse range of possible machines, varying from cell-phones to servers, that could be running agent instances. The agents connected to a

[†]`http://www.fipa.org/`

container have no restrictions on their capabilities except that must provide must provide a unique name relative to their container. This therefore creates a unique identifying name to be used for cross container communication. As each agent connects to a container, these containers provide services for that agent to take advantage of, including but not limited to allowing "each agent to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm."[2] These services are defined by in the F.I.P.A. specification titled "FIPA Agent Management Specification"[17] which provides a specification which J.A.D.E. follows that includes for basic services such as a directory facilitator, also known as a yellow pages service, and a agreed upon agent naming scheme. These directory facilitators can be connected in a hierarchy or federated together therefore allowing for a search not only under the agents local D.F. but also extending to any federated facilitators. An example from the F.I.P.A. specification of a registration for an agent named *dummy* would have the message format represented by figure 4.3.

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language fipa-sl0
  :protocol fipa-request
  :ontology fipa-agent-management
  :content
    "((action
      (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (ams-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :state active))))")
```

Figure 4.3: F.I.P.A. D.F. Registration[17]

As can be noted from this message, the information contained inside it is extensive, with the message containing specifications for ontologies, languages, receivers and message content with even more content possible. These messages and their associated format are also managed by F.I.P.A. through a working document titled "FIPA ACL Message Structure Specification"[11], whereby a standard specification for messages is defined. J.A.D.E. follows this format and all agent communication, that is through the framework occurs in this specified format. This message format allows for the following message parameters (see table 4.1) to be attached to the message, as desired with the most important parts of the message lying in its ontology, language and content sections. An ontology which is a specification for the concepts in some knowledge domain and the language that those concepts appear are the core of the message itself. The actual content which carries those concepts in some format is extremely substantial in accomplishing any activity which relies upon some data transmission accompanying the message.

Parameter	Category of Parameters
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Table 4.1: F.I.P.A. ACL Message Parameters[11]

As for a standard language the F.I.P.A. specification defined in the working document titled "FIPA SL Content Language Specification"[15] provides for three base languages and grammar sets, starting from F.I.P.A. SL0 (the minimal set) providing "the completion of an action and simple binary propositions"[15] to a much more advanced F.I.P.A. SL2 which

provides for "first order predicate and modal logic"[15] to be contained in the content of an ACL message. The content, ontology and language sections composing these messages can correspond to any string object, even serialized java objects, but most often these three sections will form one of the F.I.P.A. working groups specifications for interaction protocols[‡], thereby encouraging developers to standardize their communication patterns. As these messages are received and sent by an agent, the J.A.D.E. runtime handles the communicative activities required to deliver that message to a receiving agent. This occurs through a message transport subsystem which is also specified by F.I.P.A. in the working document titled "FIPA Agent Message Transport Service Specification"[12] which defines an abstract view of the message transport (see figure 4.4) model.

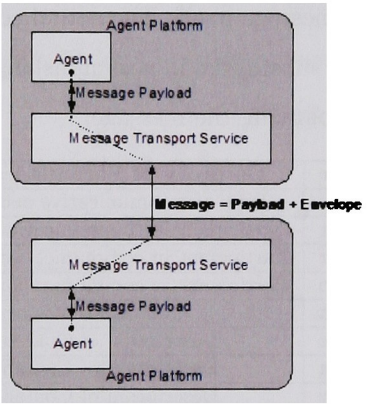


Figure 4.4: F.I.P.A. Message Transport Service Model[12]

This abstract definition of a M.T.P. allows J.A.D.E. to accommodate the actual message transmission in any method it sees fit, which as of current releases uses an R.M.I.[27] based M.T.P. for internal services and an HTTP based M.T.P. or event based transport system for agents running in inter and intra-containers respectively. These M.T.P. are extremely flexible in that they can be exchanged at runtime and can vary in type among any main or sub-containers that may exist in the platform. As was shown in [60] the J.A.D.E. platform

[‡]`http://www.fipa.org/repository/ips.php3`

does not introduce relevant overhead to the messaging system and handles scalability well for situations where agents are in the intra or inter-container categories.

The J.A.D.E. framework also provides for a unique method in which an agent and its corresponding behaviours are maintained, activated and modified. The designers of J.A.D.E. decided upon a model whereby each agent has a single-threaded[46] environment for its behaviours. An approximation of this design, shown in figure 4.5, shows how the single threaded manager runs active threads and shifts threads between active and waiting upon request (this is accomplished through a call to a *block*() method). These behaviours can be inserted or removed at runtime and are expected to only run short running tasks (or offload work to another thread) to ensure they do not block the threaded manager. To ensure this works correctly when a behaviour activates its *block*() method, the threaded manager will only reactive that behaviour upon either receival of a message (a mailbox like queue is maintained) or at a behaviours timeout request will it wakeup that behaviour. In order to deliver the messages from the mailbox, when a message is received all behaviours that are blocking are woken up and checked to see if any of them want to handle the incoming message. Thereafter the message is removed from the mailbox if any behaviour received the message or left in the mailbox for further processing by a future behaviour.

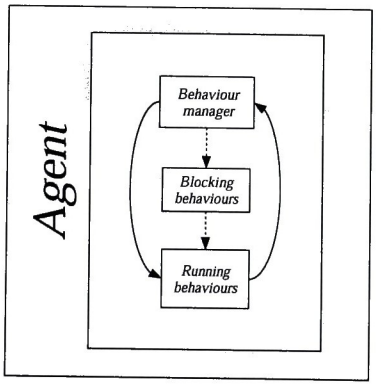


Figure 4.5: J.A.D.E. Agent-behaviour system

4.4 Joone

Although this framework is generic enough such that any classification algorithm that can be queried and trained (if desired) can be used (see chapter 5). In order to test the architecture appropriately and within reason, a subset of classification algorithms relating to neural networks (see section 3.2.2) were chosen as the algorithms for the agents in this architecture which will be performing the actual classification. To avoid having to manually implement the sophisticated algorithms that neural networks require, a open-source library named Joone was applied[38]. This library provides a modular architecture with "pluggable, reusable, and persistent code modules" for many different types of supervised and unsupervised neural network models (see section 3.2) along with a graphical editing environment.

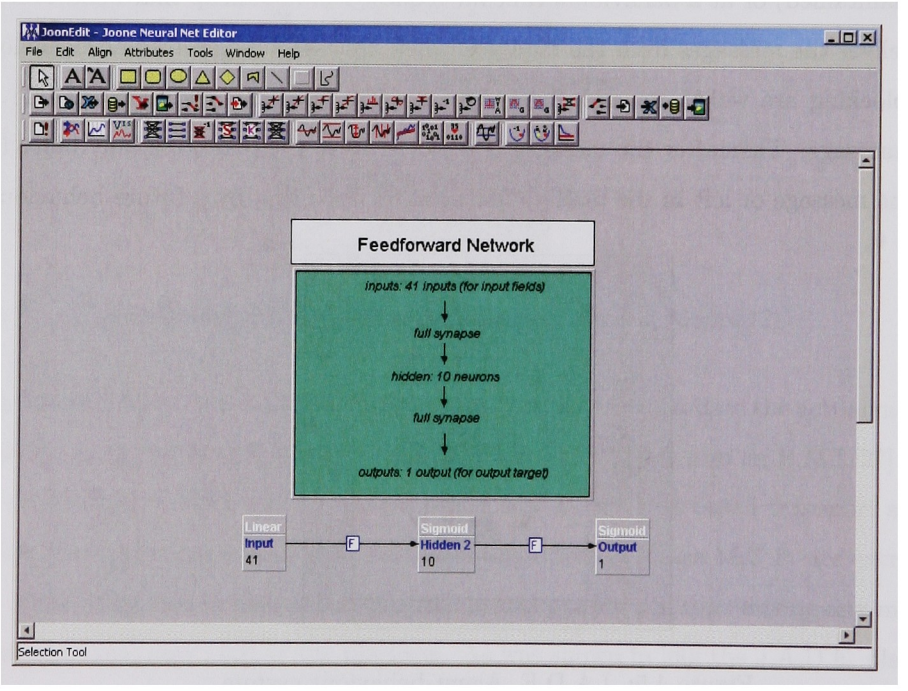


Figure 4.6: Joone Neural Net Editor

This graphical editing environment (see figure 4.6) provides for a method to manipulate and generate many different types of neural networks, therefore allowing this thesis and any future users to empirically apply a variety of different neural network structures. Although using this editor is not necessary as will be explained in chapter 5, it aided immensely in generating different types of networks without having to ensure the code for those networks was structured correctly, as this tool outputs the networks with mostly (see section 5.1.6.2) correct structure natively.

4.5 Visualization

Since one of the goals of this thesis was to provide an environment whereby novice users with limited knowledge of the agents and surrounding processes could take advantage of the potential of this framework a graphical user interface toolkit was chosen to achieve this goal. An additional benefit solving this goal provides is a relatively easy to use environment for debugging, result collection and analysis therefore aiding development and thesis experiment analysis processes.

Consequently it was chosen to use a Java provided toolkit for G.U.I. building named Swing, based off of a previous Java G.U.I. toolkit named AWT. As of Java 1.2 a set of foundation classes were created spanning the AWT, Swing, Accessibility, 2D API and drag and drop functionalities. These foundation classes were designed in order to better represent and abstract the user from the AWT design which existed previous to Java 1.2 as it provided "only the minimal amount of functionality necessary to create a windowing application[6]." The AWT also relied heavily on native code to produce the widgets (G.U.I. elements) that compose a graphical interface and was frowned upon by many developers as being a library which did not maintain the Java principle of "write once run everywhere."

The AWT and Swing combination on the other hand provides for a well designed event-dispatching framework, whereby a model-view-controller design principle[20] and corresponding abstraction layer is presented in order to allow AWT and Swing to become

a "write once run everywhere" architecture. As will be shown in the design chapter and accompanying user guide (see appendix A) the Swing framework described above and open-source libraries described below were taken full advantage of to provide an environment which satisfies the goals stated previously.

4.5.1 JFreeChart

JFreeChart, a open-source charting library providing a "free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications"[§] was used in this thesis. This library, provided under the LGPL[¶], is very extensive and provides many different charting capabilities including but not limited to scatter, pie, financial, line and bar charts. For usage in this thesis real-time scatter charts were taken advantage of for displaying clusters. Also for displaying learning error rates, when applicable, time-dependent line charts were used therefore allowing for instantaneous visual feedback on the progress of the agents performing the actual classifications.

4.5.2 JChart2D

JChart2D, another open-source charting library[‖] providing a "minimalistic charting library" under the LGPL license was used in areas where JFreeChart did not perform well. Display issues were found with JFreeChart when its charts were placed in the G.U.I. in areas where the chart was not given a lot of display area. Therefore JChart2D was used as it is able to handle limited display area problems fine. In this thesis it was used for mainly displaying resource usage related statistics such as JVM memory usage, processor usage and network traffic therefore allowing for real-time visualizations of these relevant resources.

[§] `http://www.jfree.org/jfreechart/`
[¶] `http://www.gnu.org/licenses/lgpl.txt`
[‖] `http://jchart2d.sourceforge.net/`

4.5.3 Zeus J.S.C.L.

Zeus Java Swing Components Library, an addition to swing provided many useful components and resources for the visualizations in this thesis. This LGPL library provides "solid and useful swing components"** for developers to use in their applications. Some examples of what this library provides include visual console display (JConsole), a method for displaying exceptions, errors, warnings in dialog windows (JMessage) and many other resources which make swing applications easier to program and use. These two mentioned components from this library where the main resources which were used from this library, providing for a way of viewing the console without having a console window open and for displaying messages in a manner that is much more flexible than the javax.swing.JOptionPane method.

4.6 Jakarta commons

The Apache Jakarta project, a set of projects responsible for offering "a diverse set of open source Java solutions" provides a central area†† whereby a repository of reusable Java components are located. These include well tested libraries for various design patterns, mathematical calculations, command line processing, object pooling, collections, networking and many others for various application domains. For this thesis a limited selection of these packages were taken advantage of as the extensive packages offered by this repository go beyond the application scope of this thesis. The three packages that were taken advantage of were the Commons CLI package offering command line argument parsing, the Commons Math package offering robust statistical analysis routines and the Commons Collection package providing useful collection classes.

**http://sourceforge.net/projects/zeus-jscl/
††http://jakarta.apache.org/commons/

This page intentionally left blank.

Chapter 5

Design

For this theses goals to be achieved a complete architecture (comprised of over 47,000 lines of code) had to be manifested from the ground up (with help from the technologies listed in chapter 4). This architecture is composed of six heterogenous agents, each with separate tasks, interactions, ontologies, and algorithms. In the following chapter these agents, tasks, and all aspects of design will be described. To achieve this a general overview of the architecture will be given in section 5.1, describing the high level view of all agents, tasks and interactions. Following this while be section 5.2 which will go into a greater depth on the applications, agents, tasks, interactions, ontologies, algorithms and design that compose this thesis.

5.1 Architectural overview

The general structure for this thesis starts with individual distributed, modular agents, each corresponding, collaborating and coexisting together to perform this theses goals. The global perspective of these agents and the general data flow interactions can be seen figure 5.1. This view shows the inherent distribution of each different agent type, as each agent type needs only to be connected to a J.A.D.E. container whereby they can locate the

other type of agents through the containers directory facilitator. Although the structure is distributed, as can be seen from the figure it is also partially hierarchical. This hierarchy was designed to mimic an organization and as such has agent types with similar titles and associated duties which will be discussed throughout the rest of this chapter.

Figure 5.1: High level view of the agents

5.1.1 Datasources

The entry point for data in this system is the datasource agents, acting as the primary caretaker and visualization center for data-set files [see figure 5.2]. They are also designed to be strictly reactive, as no intelligence or planning is needed to respond to inquires for record or information about those records. Their main tasks are to load data-sets from C.S.V. (comma separated values) files, converting those values if needed to numerical types, and thereafter automatically (a manual option exists) generating a training set from the converted numerical data using algorithms determined at runtime. Thereafter once these actions have commenced and these a data-sets has been loaded and a training set generated/selected the agent can become active and consequently then can respond to queries and requests for records, training sets, or information relating to those records.

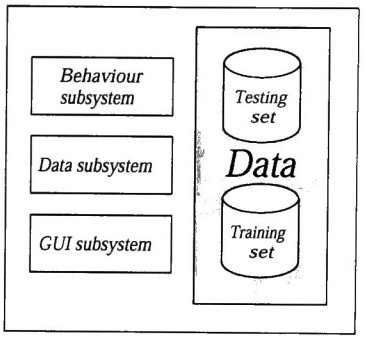


Figure 5.2: Datasource architecture

During design, these agents were created to map to the K.D.D. processes relating to selection and transformation of data, as well as data hosting, performing these tasks through crafted algorithms, behaviours and parameter defining XML configuration files. Notably left out from this procedure is the K.D.D. preprocessing stage (see figure 1.4). In order for this stage and associated processes to be accommodated in an automated fashion a separate non-agent application was designed with the general architecture for this application shown in figure 5.3. This processor was designed such that arbitrary algorithms could be activated on large data-sets, such as normalization, data cleaning, sorting, and randomization therefore accommodating the tedious K.D.D. process involving data pre-processing.

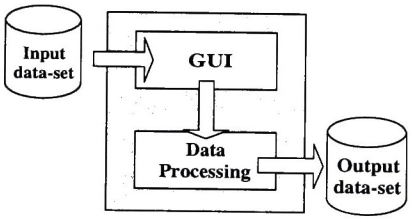


Figure 5.3: Dataprocessor architecture

As far as interactions with other agents are concerned these agents only communicate with two other agent types, these being the supervisor agents and the result reception

agents. The reason the supervisors communicate with the datasource is the following, the supervisor agents receive and apply the training set of each connected datasource to test and grade its employees on. Thereafter at a future time the supervisors may also potentially request testing records to be classified. These requested testing records will then be preprocessed by the datasource before being delivered to ensure the target class has been removed. On the other hand the interaction between the datasource and the result reception agents exists only for validation, as the result reception agent receives classifications from the collectors these result receivers then query the datasource where that record originated from to check if the classification was correct.

5.1.2 Supervisors & employees

The supervisor, being the active trainer and evaluator of employees is composed of different levels of modular subsystems as can be seen in figure 5.4, therefore represented a layered agent architecture. Even though this figure depicts the processes as rather uncomplicated the subsystems that compose this agent's architecture are in fact complex and employ a diversity of supervision and employment related algorithms (see section 5.2).

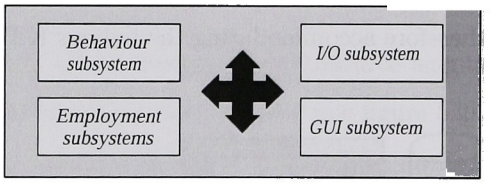


Figure 5.4: Supervisor architecture

The systems shown perform various tasks, with the G.U.I. and behaviour performing G.U.I. and behaviour running tasks and with the employment subsystems performing employment related activities. The I/O subsystem performs the retrieval of all data from datasources needed for employment activities to occur and also ensures that collector agents (up to a configurable limit) are able to connect to the supervisor for results by way of a

F.I.P.A. subscription protocol (see section 5.2).

The supervisor's employment subsystem decomposes into six employment related subsystems shown in figure 5.5. Each different subsystem handles a different task related to employment, such as grading assignments and handling training and testing processes. These subsystems also have interactions between themselves for certain cases, such as when an employee must be moved from testing to training units.

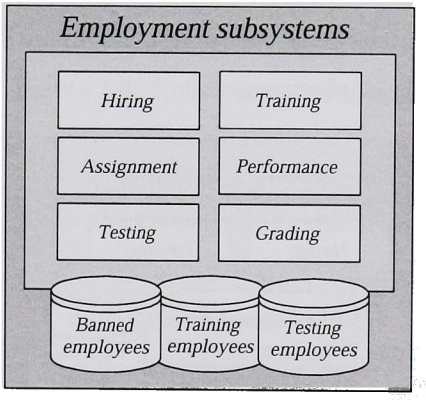


Figure 5.5: Supervisor employment subsystems

These supervisor employment related subsystems provide one of the core elements of this frameworks dynamically adapting technologies. This is because these employment related systems provide for methods to dynamically evaluate the performance of employees and their classification abilities, thereby preferring and rewarding individuals who perform "well" over individuals who do not perform "well".

The other core element of this framework is the employee agents, whose basic architecture is shown in figure 5.6. Similar to the supervisor agent this agent uses a layered architecture, therefore allowing for modularity and separation of the different subsystems this agent uses.

The top level layers are as follows, the behaviour and G.U.I. subsystem handle interaction with the G.U.I. and the J.A.D.E. behaviour subsystem while the intelligence subsystem

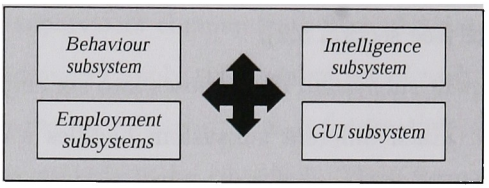


Figure 5.6: Employee architecture

handles testing, classification and training tasks. The employment layer decomposes into three subsystems with these systems being shown in figure 5.7 with each performing a task related to finding or maintaining employment.

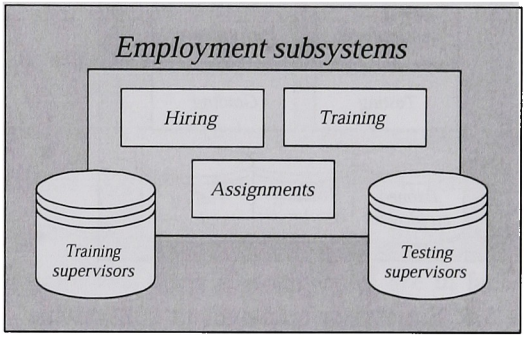


Figure 5.7: Employee employment subsystems

These set of processes and subsystems provide the secondary core of this architecture since they allow for a intelligent and adaptive responder to the supervisor's subsystems. Overall the combination of these two agents and the interactions they maintain comprises a major part of the theses architecture, as they allow for a system which can dynamically respond and adapt to changing data.

5.1.3 Collectors

The collector agents, like the datasource agents are also strictly reactive, and serve the purpose of delivering messages to receiver agents, without incurring the $1 : N$ overhead

which would occur if a supervisor directly communicated with receivers. Instead of having this $1 : N$ communication and having the supervisors know directly who the receivers are a architecture was designed to avoid this, as shown in figure 5.8.

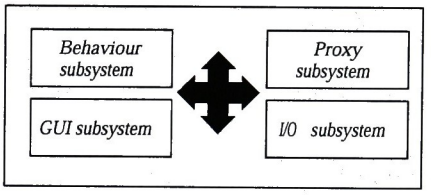


Figure 5.8: Collector architecture

The collectors are designed to enforce a level of separation between the supervisors and the individuals who would receive the classification results being made by the supervisor. It was decided that therefore a proxy subsystem, which makes up the collector would enforce this level of separation, as a supervisor would only need to send its result to a known collector, and thereby the collector would handle the delivery of that message to its receivers.

The interactions that this agent performs are the following; upon initialization this agent will receive notifications of supervisor and receiver agent initialization and it will attempt to initiate a subscription protocol for the supervisor and a subscription protocol for the receiver if possible. Thereafter the supervisor will send classification results to this agent, whom will strip the supervisor's identifying information from the messages and forward the messages to all connected receivers.

5.1.4 Receivers

The receiver agents are the outputs of the system, whereby classification results are received and displayed. They are designed be reactive and to aid in visualizing classifications received from collector agents. As is so, a big part of their architecture, as seen in figure 5.9, resides in the result display subsystem. This subsystem interacts primarily with

the behaviour subsystem and the G.U.I. subsystem in order to display useful information relating to classification results received.

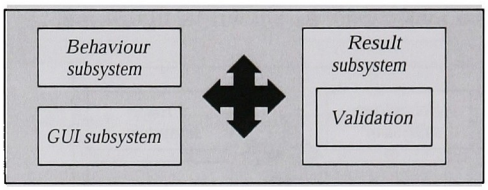


Figure 5.9: Result architecture

Another important function this subsystem performs is to attempt to identify the actual target class of a received classification, performing this through verification with the records originating datasource. Thereafter once this verification and associated actual target class is received, the receivers current visualization environment will display this information using various methods, with each method designed to aid in the different aspects of the analysis process.

5.1.5 Monitors

The group of monitor agents exist in this thesis for visualizing information related to agents activity (most importantly whether they are alive or dead as J.A.D.E. does not automatically remove abnormally terminated dead agents). As can be seen from figure 5.10 these agents implement an architecture which monitors other agents using whatever algorithms are provided to enable this monitoring process.

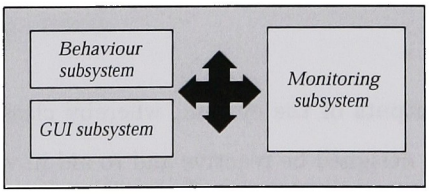


Figure 5.10: Monitor architecture

The interaction these agents have does not depend on any agent type, but depends instead on a specified ontology. This ontology, which will be described in depth in section 5.2.3, allows for responses to inquires about the whether a certain agent is alive or not alive (possibly due to failure). This agent uses this ontology to periodically query whether agents implementing this ontology are alive or not alive, thereby allowing for this information to be displayed in the previously discussed G.U.I. subsystem.

5.1.6 Applications

As this framework is rather large, a number of auxiliary applications were implemented to aid in the frameworks processes, besides the data processor already mentioned. Consequently these applications architectures will be discussed in the following section, starting with an application created to aid in starting agents and their associated containers and ending with a utility created to repair a problem found in the joone G.U.I. editor.

5.1.6.1 Agent starter

This application was designed to aid in the launching of agents and their associated containers. As it was found early in the development of the thesis the starting of each individual agent was command-line based and extremely tedious. Therefore to avoid this tedious process a centralized application and associated architecture was designed to aid in the agent and container launching process, as can be seen in figure 5.11.

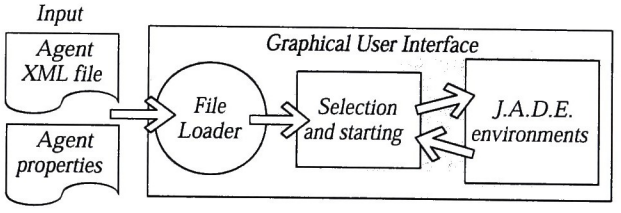


Figure 5.11: Agent starter architecture

This G.U.I. application as can be seen from the figure reads in the parameters needed to start a J.A.D.E. agent from a specified XML file (along with a global agent properties file) and allows for selection and starting of the agents using these parameters and associated properties. Once an agent is selected in the G.U.I. it can then be further configured and launched into a existing or new J.A.D.E. container. Overall this architecture enables the launching of J.A.D.E. agents to be achieved in a straightforward process, avoiding the tedious command-line process which preceded it.

5.1.6.2 Neural network repair

The joone G.U.I. editor was found to be faulty in a certain limited area. Upon export and corresponding serialization of the neural network from the editor, it would not set the appropriate input and output layers of the neural network. Unfortunately the joone runtime engine needs these layers specified and if not found the engine will attempt to locate them. This would be normally be fine but the engines algorithm does not always locate the correct layers.

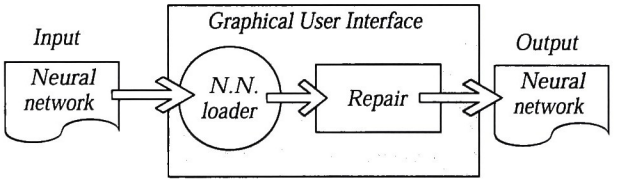


Figure 5.12: Neural repair architecture

Therefore a small G.U.I. application and architecture as can be seen in figure 5.12 was developed to enforce the specification of these layers, bypassing the malfunctioning joone algorithm. This application as can be seen from the diagram takes in a serialized network from the G.U.I. editor and applies the necessary repairs. Thereafter the application outputs a new repaired neural network which can then be used by the employee agents for classification processes.

5.2 Implementation details

5.2.1 Applications

5.2.1.1 Data processor

This application, as mentioned previously was designed to accommodate the K.D.D. process involving data preprocessing and cleaning. This involved the creation of a G.U.I. and different implementations of algorithms corresponding to data-set preprocessing. The G.U.I. application (see figure B.16 in appendix B) was designed using swing with actions being activated through a separate threaded executor[46] thereby avoiding locking up the interface. The G.U.I. itself allows for loading of previous or new zip files containing data-sets, selecting an action to occur on that file and selecting an output directory where any new contents should be written to (see appendix A). The important part of the implementation for this application involves the processing algorithms which were chosen to aid in the processing of large data-sets.

The algorithms composing these actions, each take in a set of zip-files, with each zip file potentially containing multiple data files inside of it. These algorithms, shown in the following paragraphs, analyze these zip files, using the Java libraries for streaming zip file reading/writing and process their contents (C.S.V. files) in an arbitrary manner therefore resulting in information about those zip files or a new set of zip files. These individual algorithms all derive from a base interface, this being shown in figure 5.13 with the implemented actions, and they will be discussed in the following sections.

Find largest-smallest: This algorithm implementation parses contents of the zip files, and identifies the columns of the C.S.V. files and determines the largest and smallest numerical values of those columns. This happens across all the files in the zip files and is useful in determining statistical information relating to the data-sets.

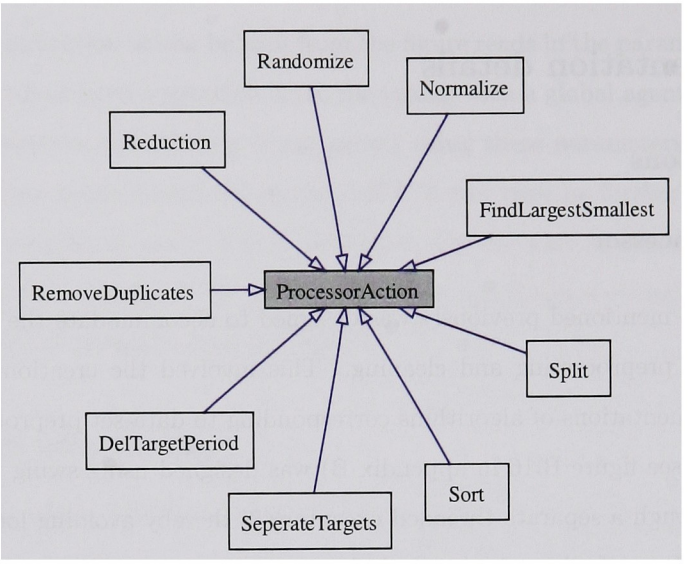


Figure 5.13: Processing algorithms

Normalize: This algorithm, uses the "find largest-smallest" algorithm as a precursor, to determine the columns in the data files across the zip files with the largest and smallest values. Thereafter it scale the values into the positive range and normalizes the values to lie in the $[0, 1]$ range. This algorithm is useful as many different algorithms, neural networks especially, require the data be in a specified range.

Reduction: This algorithm performs a rather straightforward function, that is to reduce the number of records in a data-set to a specified amount. Therefore to accomplish this each data-set is iterated over and new data-sets are outputted with the desired number of records, if that many records exist, otherwise the original data-set is duplicated.

Delete target period: This algorithm was designed for a single data-set type of data-set record format. A certain number of data-sets have formats that specify the record should end with a "." but this was not needed for the data-sets used by this architecture (mostly due to personal aesthetic reasons). Therefore this algorithm iterates over all records and

removes the trailing period if it exists.

Separate targets: This algorithm, works in combination with a fields XML file, which notable contains a record which identifies the target column of any data-set C.S.V. file being read. This algorithm will then use that column and break up any data-sets such that they only contain a single target class per output file. This algorithm is useful for testing and training since using it before launching the platform allows for various combinations of N different target classes to be imported as data-sets.

Sort: This algorithm implementation attempts to sort the contents of the zip files. This occurs through a unique sorting method common to database technologies, since this required when sorting large files which can not be sorted in memory. The algorithm implemented is a derivation of the two-phase multi-way mergesort[21], an algorithm which extends the mergesort algorithm to allow for out of memory sorting of large files. Using this algorithm allows for a method whereby large files can be sorted by splitting the large file into N sorted files and then selecting records from those N files in sorted order. These sorted records are then iteratively outputted to a new output file until all N sorted files are depleted.

Remove duplicates: This algorithm is designed to be activated after the sorting algorithm, as this algorithm iterates over all records and checks if a following record is the same as the current record. If that record is different it is outputted and it is then used as the checking record, otherwise the record is not outputted and the iteration continues.

Randomize: This algorithm applies a randomization method to large files. This algorithm, since large data-sets may not fit into memory splits those files up into N temporary files each containing a maximum of M randomized records. Thereafter it will iteratively output a record from a random temporary file until all temporary files have been exhausted.

Split: This algorithm applies a splitting method to large files, thereby allowing a zip file containing a large data-set to be split up into N smaller data-sets. This algorithm works by iterating over the zip files provided, going through each zip file entry in the zip files, and outputting a new set of zip files with new entries being created to match the desired number of records.

5.2.1.2 Agent starter

This application (see figures B.17, B.18 and B.19 in appendix B) was implemented using a combination of techniques, each contributing to the goal of making agents easier to launch and monitor. In order to load the global properties file (all agents get their configuration from this) and the agent XML file (the G.U.I. uses this to populate which agents can be started) from the command-line, the apache CLI package was used. This allowed for specifying the options in a GNU format, thereby maintaining a consistency among command-line options for users familiar with many of the thousands of GNU applications. Secondly the XML files were then read-in via java's properties reading methods and a custom XML SAX parser designed to read-in a custom XML with a specific schema for defining agents and their associated properties.

In order to accomplish the actual activating of the agents, methods were implemented which interface with the J.A.D.E. packages. These methods ensure that agents and containers are created correctly and succeed in their activation. In order to perform this, as swing is single threaded, a separate thread is used to create agents and containers, thereby avoiding locking up the swing thread. Any results that are sent back by this thread to the G.U.I. are always channeled through the correct methods to ensure that the swing thread does not behave incorrectly. Thereafter once an agent has been created, an implementation class will then be assigned to handle notification of that agents "death", which also creates an update in the G.U.I. as all active agents are displayed there.

Also implemented in the G.U.I. is a visual representation of the current JVM memory

usage where the application is running. This was also done using a separate thread of control, whereby the thread is activated after a certain period and updates the memory usage which is shown in a chart instance. Along with this, the G.U.I. also implements a visual representation of the standard console environment. This helps in avoiding the need to launch an monitor the console where the Java application was started since the implementation underlying the G.U.I. console redirects the output of the original console to the G.U.I. environment.

5.2.1.3 Neural network repair

This application as discussed in the architecture section, applies a rather straightforward process for implementation. To implement this application a collection of swing objects were combined together to provide for a clean implementation of a G.U.I. environment (see figure B.20 in appendix B). These G.U.I. objects specify where the loading of the defective network should come from, where the output network should be placed and the input and output layer names. Once a specific button is clicked, the values these objects contain will be used to set the specific layers of the network, if possible, thereby completing the repair and outputting a corrected network in a serialized form.

5.2.2 Shared agent components

As the agents share many common features, it was decided during implementation that they should share a common interface and common base class. This common base class (which implements the shared agent interface) can be seen in figure 5.14. One of the benefits of this design is that as each agent to share a common base class, this allows for a central entry point for future expansion of this framework, as well as reducing code redundancy in the current implementation.

It also and more immediately provides a shared set of methods for all agents to use, including methods and classes to enable a centralized point for G.U.I. events and corre-

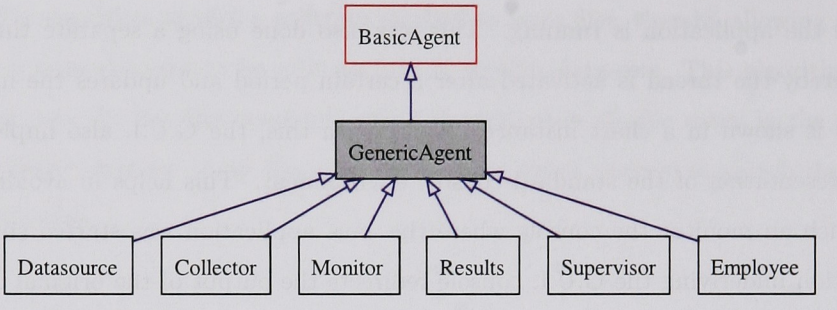


Figure 5.14: Agent base

sponding agent events along with various other useful methods thereby encouraging a clean separation of the model from the view. The important part of the implementation of this parent class is the handling of these two types of events. In order to handle these events

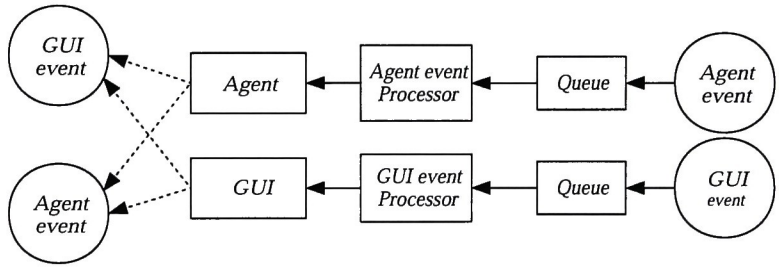


Figure 5.15: Event subsystems

in a well thought out manner an implementation which uses two separate threads of control to handle events going between the agent and the G.U.I. environments was decided upon. These threads receive the events through channels and blocking queues, common synchronization implementations for abstractions of a "mailbox." The process works as follows, once an event is received in the mailbox the receiving thread will wake-up and handle that event. If it is aware of that event type it will then activate an action in the agent or by activating an action in the G.U.I. which therefore may cause further events to be fired. This pattern follows a well known design pattern called the mediator pattern,

whereby this pattern "allows groups of objects to communicate in a disassociated manner and encapsulates this communication while keeping the objects loosely coupled."[34]

Another shared feature used by all agents is a shared G.U.I. panel, an extension of the java JPanel class. This abstracted panel implementation contains standard methods for displaying logging data inside of the G.U.I. along with methods to retrieve and activate the individual thread for G.U.I. events. This is the G.U.I. event receiver thread defined previously which will be customized by each inheriting class, as each agent panel needs to receive different types of G.U.I. events and handle them accordingly. Certain G.U.I. events

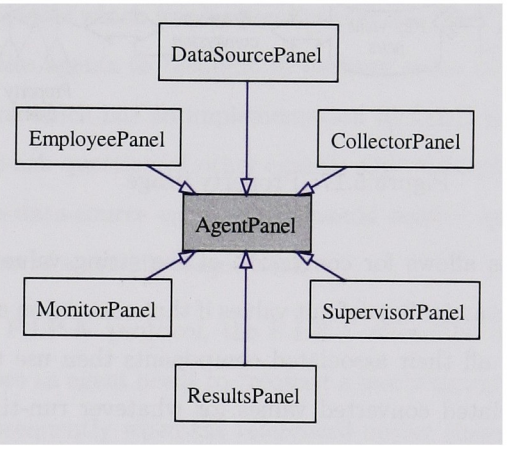


Figure 5.16: Agent panels

though are shared by the entire group, as each agent has events and corresponding G.U.I. components for displaying logging information and displaying messages which are sent or received. Two other shared component which use these messages are two charts which display the bandwidth usage created by the send and receive of these messages in a visual format (see figure B.15 in appendix B). A final shared G.U.I. component is an "about" dialog which each agent shares, which displays runtime environment properties, agent properties, active behaviours and various other useful information about the environment the agent resides in (see figure B.14 in appendix B).

5.2.2.1 Configuration

The configuration of the agents, and all run-time properties used by those agents, as discussed partially in section 5.2.1.2 was implemented by taking advantage of the Java properties implementation. This implementation allows for reading of key-value pairs from XML or text files. As XML is used throughout this platform the XML property reader was chosen as the implementation, with a custom class defined to convert the value into a meaningful object.

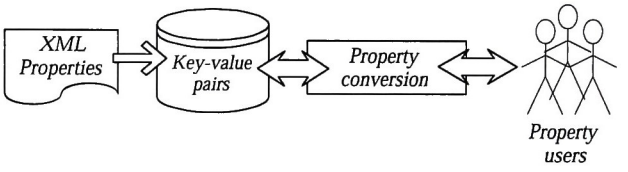


Figure 5.17: Property usage

This conversion process allows for conversion of the string value into all of the java primitives objects as well as assigning default values if that conversion can not be completed correctly. The agents and all their associated components then use this conversion class to retrieve keys and associated converted values for whatever run-time parameters they require (as seen in figure 5.17). This whole process therefore avoids statically defining those values in the implementation which would thereby require source code modification for any parameter changes.

5.2.2.2 Protocols

Certain protocols used for communication activities are shared among the whole group of agents that this framework defines. These include the standard F.I.P.A. interaction protocols along with designed protocols which are not available in the J.A.D.E. packages. Therefore in the following section these protocols and their implementation will be discussed, giving an overview of the processes these protocols encompass.

As far as F.I.P.A. protocols are concerned, three protocols were taken advantage of to enable interagent actions to be accomplished in the implementation of this thesis. One of the protocols which was used frequently was the F.I.P.A. request interaction protocol[14] which allows "allows one agent to request another to perform some action.[14]" This protocol, with a default implementation created by J.A.D.E., was taken advantage of multiple times to perform various request-like activities. One example of where this was implemented was when the data-source agent gets requested to send data records to external individuals.

A second F.I.P.A. protocol which was frequently used was the F.I.P.A. query interaction protocol[13] which enables agents to "request to perform some kind of action on another agent.[13]" This protocol which has an implementation in J.A.D.E. was used when agents in this platform need to ask question of other agents. One example of an implementation of this was through the data-source agent which would receive queries to determine if a given record instance was of a certain target class.

The third and final F.I.P.A. protocol, the F.I.P.A. subscribe interaction protocol[16], was applied in areas where an agent needs to "request a receiving agent to perform an action on subscription and subsequently when the referenced object changes.[16]" This protocol, also with a default implementation in J.A.D.E., was used by the supervisor, collector and receiver agents as a method to subscribe to each others messages. Another area where this protocol was used is subscribing to the directory facilitators to receive information about agent creation and deletion. This is available due to the directory facilitators in J.A.D.E. implementing "the fipa-subscribe interaction protocol in order to allow agents to subscribe for being notified about registration, deregistration and modifications of certain agent descriptions.[17]" In order to take advantage of this, a implementation was defined in this architecture in which an agent could specify just the agent description they desired, or specific ontologies and they would therefore be notified of an agents creation or deletion with that description.

A separate protocol was also generated to enable agents to respond to inquiries about whether they are alive or dead. This protocol implemented a liveness test using a designed protocol which can be seen in figure 5.18. This protocol was designed due to the fact that J.A.D.E. does not implement a timeout mechanism to unregister agents from the directory facilitator if they have stopped responding. With this mechanism agents who attempt to contact non-responding agents will be able to take some type of action if the receiving participant does not respond within the n millisecond timeout provided by the protocol.

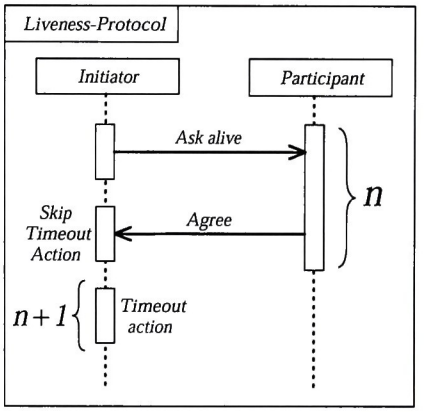


Figure 5.18: Liveness protocol

For an agent to implement this protocol, it was decided that they need to only list themselves as having the liveness ontology, which acts as an interface definition allowing other agents to know that the agent can respond to inquires. At a behaviour level, an always on behaviour is added to each agent which will implement this ontology. It will receive queries asking whether the agent is alive and if so the behaviour will respond with an liveness ontology specific "agree" message. In order for a initiator to take advantage of this protocol, the chain of responsibility design pattern was taken advantage of to enable various behaviours to first check if the participant is alive and then proceed. This was implemented by forming a chain, whereby the first action in the chain would be to initiate the liveness protocol and ensure the individual is responding. Then the chain would continue to to be

unraveled whereby whatever original action the behaviour defined would then occur. If the agent does not respond, the chain will not be unraveled and the agent will be notified that the chain did not unravel as it should have.

5.2.2.3 Behaviours

There are only two behaviours which are shared among all the agents these being the liveness responder behaviour which performs the role of the participant of the liveness protocol and a liveness "manager" behaviour. The liveness "managers" sole existence is to receive messages specifying individuals to check for responsiveness. Thereafter it will initiate asynchronously a child behaviour who thereby will attempt to check the liveness of the desired individuals (acting as the initiator in the protocol). Once a response is received from this child behaviour this result is then forwarded to the component who initiated the check, which then can take any corresponding action to deal with the response.

5.2.3 Ontologies

In order to accommodate external communication between agents and internal communication between an agents own behaviours, several different ontologies were defined and used for the messages which are transmitted. The reason for the separation of internal and external ontologies is the fact that J.A.D.E. does not provide for a way for intra-agent behaviours to communicate in an asynchronous manner. Although using shared memory regions is possible this does not allow the behaviours to take advantage of the blocking nature of the J.A.D.E. behaviour subsystem without reimplementing many of the principles the subsystem already performs. Furthermore having this separation enables non-behaviour code to communicate with behaviour code and vice-versa as any intra-agent code can now deliver messages to running or blocked behaviours.

Therefore these two groups of ontologies were implemented, by extending the base J.A.D.E. ontology and then retrieving the ontologies using the singleton pattern to ensure

only one instance of the ontology exists and the factory pattern which helps eliminate the coupling between instantiation and application-specific code. Thereafter the retrieved ontologies allow for agents to ensure they are using the same meanings for the content in their messages when communicating with another agent who supports that ontology or communicating across behaviours. Subsequently in order to describe these ontologies, in the following section the relevant concepts of these ontologies shall be described along with the agents who use them.

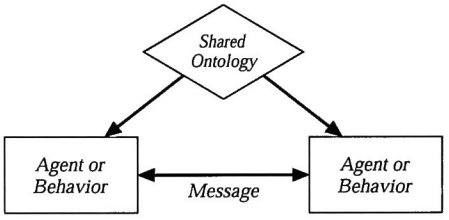


Figure 5.19: Agent-ontology interaction

5.2.3.1 External ontologies

Datasource: This ontology defines concept related to requesting records, verifying record types and queries related to those records. This ontology is used by the datasource, supervisor and reception agents as they all need to know how to receive and send records and learn various information about those records.

Supervisor: This ontology defines concepts for requesting employment information, confirming and disconfirming employment along with concepts for employees to use to request testing and training units. This ontology is used by the supervisor and employee agents as both need to have the ability to send messages with a shared meaning to allow for the correct operation of supervisor related activities.

Employee: This ontology defines concepts for employee related activities. This includes concepts relating to records, performing tests and predictions, receiving training units and receiving the results of tests and predictions. This ontology is shared between the employee and supervisor agents as both need to communicate these concepts to allow for employees to perform their corresponding duties.

Subscribe: This ontology is used for agents who are implementing the F.I.P.A. subscription protocol and are sending classification results to participants. This ontology defines the meanings of concepts which are used to send these results, which come from subscribing, in a structured manner. This ontology is used by the supervisor and collector agents as both transmit or retransmit classification results.

Liveness: This ontology contains no definitions of concepts as it only acts as a marker that the messages which come in marked with this ontology will be responded to with a message of the same ontology with an "agree" performative (if that agent is responsive). Therefore no actual content and associated concepts need to exist for this to be achieved.

5.2.3.2 Internal ontologies

Proxy: This ontology defines concepts relating to checking if it is possible for an internal behaviour to proxy. This was implemented since in order to send a response to an external agent as to whether they connect to the receivers proxy an internal set of checks must be first done to ensure this is possible. This is therefore used by the two agent which need to perform these tests, that is the supervisor and collector agents.

Liveness: This ontology describes concepts which allow internal behaviours to query a internal liveness "manager" who will then perform the activities necessary to see if a participant is responsive. Thereafter once this process finishes, the internal manager will respond back to the invoking internal behaviour as to the results of the liveness query.

Supervisor: This ontology defines concepts used by the supervisor agent to request an internal data "manager" to get records from a datasource, whether this be testing records or training records. This abstracts the process of getting records from any implementing behaviours and allows this internal manager to handle all retrieval of records for the supervisor to then use in its processes.

Subscribe: This ontology defines which are forwarded to an internal subscription "manager" who then will respond as to whether an individual can be subscribed to a receiving agent or whether they can not be subscribed. This is used by the by the supervisor, collector and reception agents as all three of these agents are involved in subscription processes and need this internal manager to check if a subscription is possible.

Checking: This ontology defines concepts used by the result reception agent to off-load checks of a records type to an internal "manager" thereby allowing for these checks to be made asynchronously with the actual type being received sometime in the future.

5.2.4 Datasources

5.2.4.1 Data

In order to retrieve data from zip files, with these zip files containing C.S.V. files an implementation was built which followed the processes shown in figure 5.20. To accommodate the conversion of zip files, each containing CSV files, into in memory records the Java capabilities to stream zip files were taken advantage of along with threaded reading to avoid G.U.I. lockup. In order to read these C.S.V. files an open source parser for C.S.V. files was used, which offered a robust method for reading C.S.V. files using a lexical parsing system built by JFlex*. Subsequently once the "unconverted" record data is read into memory the records will be "converted" to numerical values. This conversion process is

*http://jflex.de/

guided by an XML file which specifies which columns need to be converted and a mapping of the nominal values in those columns to numerical values. Following this, the G.U.I.

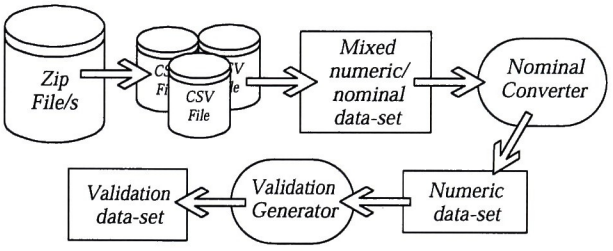


Figure 5.20: File data flow

displays both of these data-sets and if an option is selected in the G.U.I. an auto-validation process will initiate. The validation data-set generator is abstracted so that it receives the "converted" and "unconverted" data and returns a new data-set which will be used as the validation data-set. In the current implementation an instance is used which will take a certain percent (this is configurable up to 99%) of records from the "converted" data-set and use those records as the validation data-set. Thereafter the algorithm will remove the records it has retrieved from both "converted" and "unconverted" data-sets thereby ensuring these records are not used in any further classification processes.

5.2.4.2 Behaviours

In order for this agent to have a meaningful purpose, certain behaviours are by default initiated when the agent is activated (data must be loaded first for this to occur). This agent responds to query responses, following the F.I.P.A. protocol, as well as the request protocol, also following the F.I.P.A. protocol. The query behaviour is used to respond to questions asking whether instances exist along with queries to determine a records actual target class. Similarly the request behaviour is used to respond to requests for data-set related resources. These include responses to requests for data-set information, random or specific records and validation set records. These two behaviours are always-on as this

agent is reactive and simply continually wait for messages to react to.

5.2.4.3 G.U.I.

In order to have a meaningful G.U.I. which displays useful information a implementation was chosen which would display the data-sets and related information in various tables (see figure B.2 and figure B.3 in appendix B). Using extensions of the JTable class along with table models the data-sets being used by this agent can thereby be viewed in a tabular format. This agent also populates additional tables related to statistical (average, sum, variance, mean, max, min) information about the two numerical data-sets. In order to visualize information relating to behaviour activity, certain G.U.I. events are received which inform the G.U.I. when records are sent, thereby highlighting that record in the tables. Other similar events are received which inform the G.U.I. of each behaviours current activities thereby allowing for visual feedback of the agents underlying processes.

5.2.5 Supervisors

5.2.5.1 Employment

The hiring subsystem is the initial point whereby employees who wish to be hired are channeled through. Initially a hiring algorithm occurs, which starts a two-phase asynchronous process which involves initially telling the employee about the supervisors current state (how many current employees and how many are allowed). Once this information has been sent the supervisor may then receive a request to confirm that employees employment. If the supervisor is still accepting employment offers a "confirm" message will be sent back, otherwise a "disconfirm" message will be sent.

Once the hiring algorithm finishes with a "confirm" message thereby confirming the employee is hired, the hiring subsystem then off-loads that employee to the testing subsystem. Consequently the testing subsystem asynchronously initializes a dynamically generated test for the newly hired employee (if validation data exists, otherwise the employee is put in a

backlog) to determine the employees current knowledge. When validation data is received (after the supervisor connects to a datasource) the backlog will be emptied and the employees will be channeled through a test generator. The test generator uses an interface implementation which as of the current implementation takes a random amount of records from one of the N validation data-sets the supervisor currently contains and sends these records (stripped of their target class) as a message to the employee to test on.

When this test result is received it is off-loaded asynchronously to the grading subsystem for evaluation (if the result is not received the employee is fired and banned for a configurable time period). The grading subsystem then uses a test evaluator which returns a grade between 0 and 100. The algorithm used to achieve this is rather straightforward and compares the results received by the employee with the actual target class and applies a configurable numerical tolerance to allow for variations in the actual and received value. Thereafter once the employee has been graded, a configurable passing grade will be checked which will determine whether the agent is placed in the training or testing groups. Upon completion of this test and subsequent grading the employee will be sent a message telling whether they have passed or failed, thereby allowing for the employee to perform the necessary tasks to improve if necessary.

Once an employee has been tested, and if they pass, they will then be moved to the assignment subsystem whereby thereafter classification assignments maybe initiated with data from the I/O subsystem (if testing data exists). These classification assignments will be asynchronously initiated when a configurable number of employees are active in the assigning group, although this will not stop classification records from being requested as they will be put into a back-log for future classification. Once the specific number of testing employees have been achieved assignments will begin at a configurable rate (using a separate assignment starting thread), whereby the record to be classified will be sent to all employees in the testing group and a separate thread will wait for the classification responses. If no responses are received within a given timeout the classification record will

be placed back into the back-log for future classification.

If at least a minimum number of responses are received a group consensus algorithm is activated in a separate thread to determine the individual assignment grades and the record target class (as target class values could vary among the results received). To determine the target class of the assignment, the k-means algorithm applies the euclidean or manhattan distance formulas to calculate the distance between cluster and non-cluster points which represent a result of the assignment.

$$\text{Classification result } P = (p_1, p_2, \ldots, p_n)$$
$$\text{Cluster centriod } Q = (q_1, q_2, \ldots, q_n)$$
$$\text{Euclidean} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$
$$\text{Manhattan} = |p_1 - q_1| + |p_2 - q_2| + \ldots + |p_n - q_n|$$

To achieve this a single cluster is initiated and the k-means algorithm runs for a specific number of iterations to determine the centroid of all classification results received. This centroid is then what the supervisor will consider as the target value of all the given responses, and it will then use this centroid point to grade the performance of the employees who gave responses for that assignment. This grading is achieved by using the employee's result and the associated data point this creates and comparing the distance between the result and the centroid. This distance will then yield a numerical value which is then processed to produce a grade for that employee which as of this implementation takes the maximum value the distance can be (usually 1.0) and subtracts it from the actual distance to yield how close the employee's result was from the centroid. This distance is then multiplied by 100 to yield a grade which will then be compared to a configurable assignment passing grade to determine if the employee passed or failed. If they fail, the employee will be notified and moved back to the training group and told to retrain until they pass another test. Whether they pass or fail though this grade will be noted in a grade history until a performance review occurs, thereby making long term performance

analyzable.

In order to make the results of classifications useful the centroid result is compared to a mapping of target class to numerical values which accompanies the records in the supervisors data-set storage units (each connected datasource sends this mapping). The nearest target class, applying euclidean distance, to the centroid is therefore chosen as the actual target and that target class is then delivered to any connected collector agents. The collector agents will then benevolently deliver that target class (along with the originating instance and datasource) to any receivers down the line thereby completing the classification of that record. Simultaneously the classified record is placed into the local validation data-set that the supervisor maintains for that datasource, thereby allowing for the record to be used in future training and testing.

Concurrently an evaluation also occurs periodically when assignments are graded to give an indicator of long-term performance. This indicator compares a past history of test grades, performance and assignment grades to determine how well the employee is doing over time (the amount of past histories to keep is modifiable). As of the current implementation, the algorithm used to achieve this is to average the grades of each section and weight those grades using a configurable weight for each value (applying equation 5.1).

$$pgrade_{n+1} = \frac{\sum_{i=1}^{n} aweight * agrade_i + \sum_{j=1}^{n} tweight * tgrade_j + \sum_{k=1}^{n} pweight * pgrade_k}{\sum_{i=1}^{n} aweight_i + \sum_{j=1}^{n} tweight_j + \sum_{k=1}^{n} pweight_k} \tag{5.1}$$

Thereafter this grade is compared against a passing grade (different from previous passing grades) to determine if this employee needs to be moved from the testing to training group. Another procedure to evaluate the overall performance of the employee is the following; after k classification assignments a retest will be initiated whereby the employee will be sent a retest to evaluate how the employee is adapting to the data being classified.

As for these employees that fail either a test or a performance evaluation they will be placed into the training subsystem and will remain in that subsystem until they pass

a test. Until they do pass a test, the training subsystem will asynchronously respond to future requests, including requests for training data and requests for retesting. When a retesting is received testing subsystem will initiate a test, and the process will start over again. When training data is requested, a random validation set is chosen and a percentage of this validation set is sent as a message to the requesting employee (with target classes intact).

5.2.5.2 Behaviours

In order to accommodate the complex processes occurring for employment certain behaviours are always-on or added during runtime to ensure the employment processes are fluid and responsive. As the employment processes and associated classes are not themselves behaviours, these behaviours are only used to interact with external agents. To allow this to occur, it was decided that during implementation a set of forwarding classes would be put in place which would receive the various messages from connected agents and notify the corresponding employment subsystems. Therefore a minimal number of classes exist as behaviours for the supervisor agent, as messages are forwarded to there related subsystem whom will then deal with the actual content.

One situation where the forwarding class structure is not used is the data retrieval behaviour which connects to data-sources and receives validation data-sets and requests records. In order to accomplish this behaviour the internal messaging system was used to allow for the I/O subsystem to send internal messages which a running data "manager" behaviour would receive. Subsequently this manager would spawn a child behaviour who would asynchronously attempt to retrieve the validation set or get records from the specified datasource. Thereafter upon success or failure the child behaviour will tell the "manager" the results, which if data was received would be placed into the supervisors local mirror of the datasources data-set. These data-sets were implemented to be observable (following the observer pattern) so that employment subsystems would be notified when data is inserted.

This is required and used by the testing and assignment subsystems as they have backlog's which could be unloaded when new data is received.

The employment subsystem also keeps an active behaviour which periodically queries the liveness manager to check the responsiveness of connected employees, datasources and collectors to ensure they are still alive and responding. If they do not respond, they will be disconnected and removed from the employment systems. Additionally if the non-responsive unit is a employee they will be fired which will then potentially affect other aspects of the employment subsystems (assignment starting in particular).

In order to connect to collectors, a subscription responder behaviour is always kept online. This responder follows the F.I.P.A. subscription protocol and as such receives requests asking for subscriptions, which may or may not be confirmed due to a subscription limit which the supervisor maintains internally. If a confirmation occurs, then the supervisor will thereafter forward classification results to the collector whenever classification assignments are completed or until the subscription is canceled.

5.2.5.3 G.U.I.

In order to develop a useful G.U.I. for this agent it was decided that the implementation would allow for viewing of the current assignments, employees, collectors, grade remarks, k-means analysis and the datasource data-set units received (see figure B.4 and figure B.5 in appendix B). To accomplish these visualizations the G.U.I. receives observation events (following the observer pattern) to tell it when these items are added, changed or removed.

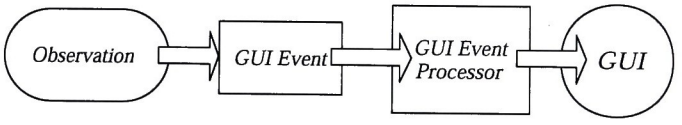


Figure 5.21: Observation-event process

This was accomplished by implementing the observable pattern on the employee, collector and data-set storage classes thereby allowing the G.U.I. to receive notification when items are added or removed and take corresponding actions to show the originating data in the interface. In order to receive grade remarks, an event is fired from the grading system which the G.U.I. receives and then adds that remark to a JTable instance. This thereby allows for the grade, the employee who received that grade and the remark (performance, assignment, testing) type to be displayed in the interface which allows for visual progress monitoring of evaluations.

Finally in order to visualize the ongoing k-means analysis, thereby allowing for a visualization of the current assignment grading process the k-means algorithm notifies a panel which contains a JFreeChart instance when the k-means algorithms adds, moves or removes cluster and data points. This instance will then display these points in a visual manner thereby allowing for a visualization of the assignment grading process.

5.2.6 Employees

5.2.6.1 Intelligence

In order to allow for any intelligence algorithm to be used in this agent a design was chosen whereby a set of interfaces is all that needs to be implemented by any intelligence algorithm. These interfaces (seen in figure 5.22) define methods to retrieve predictions on multiple/single records and to perform training on records which have there target class intact. Therefore as an intelligence implementation needs to only be able to perform these two types of operations this allows for future expansion of this framework to potentially test many different intelligence algorithms.

As it was mentioned previously though the employee's in this implementation use joone and the associated neural networks made available by this library (after layer repairs have been performed). Thereby when these networks are loaded through the G.U.I. they are encapsulated in a set of classes which implement the interfaces described previously. This

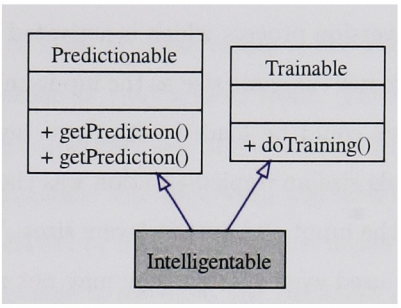


Figure 5.22: Intelligence interfaces

allows for any neural network to be loaded, supervised or unsupervised and with any type of structure since the encapsulating classes have been implemented only to be concerned with the training and testing processes and not how the neural network is structured.

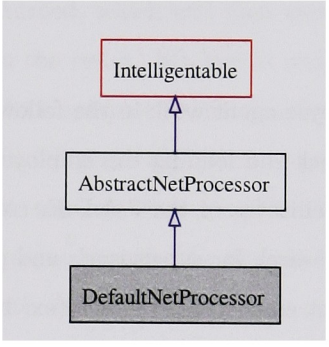


Figure 5.23: Neural network processors

These two classes, as can be seen in figure 5.23, are directly related to the intelligence interfaces defined previously. The *AbstractNetProcessor* performs the associated duties related to performing predictions and handling training processes. The *DefaultNetProcessor* extends this abstract class and provides implementations of abstract methods for retrieving the neural network learning rate, total epochs and other neural network parameters. It also handles the saving of the neural network as well as sending events to the G.U.I. when the neural network error rates change.

One last important procedure handled by the *DefaultNetProcessor* class is a set of

methods which implement a conversion process which is activated when input records and associated target class records are not the same size as the input and output neural network layers. Since any neural network could be loaded with these layer sizes potentially not matching the classification records size an implementation was chosen which would shrink or zero fill the records to match the input and output layers sizes. This therefore allows for neural network structures to be used even though they may not match the records being classified, since the scaling process will ensure the record is always the correct size. One caution must be stated though as information may be lost in the implemented shrinking process, since to reduce the size of a record the trailing end of the record is clipped to fit the matching layers size.

5.2.6.2 Employment

The actions performed by an employee agent work in the following manner. Upon initialization and successful intelligent back-end loading the employee agent will go online and register itself with the directory facilitator of the J.A.D.E. container it is connected to. Thereafter the employee agent will search for supervisor's and query those supervisors for information relating to their current employment status (ex. how many other employees are connected).

Subsequently a asynchronous two-phase employment process will be used which works through the following procedure. When information is received about a supervisor it is placed into a back-log of limited size as the first step of the overall process. Thereafter a separate thread will periodically activate and will check if the employee wishes to be employed by more supervisors. If this is the case, the back-log will be analyzed and a supervisor from this backlog will be selected. The current algorithm for selecting this supervisor sorts the information about the supervisor (the number of current employees and the maximum possible number of employees) in increasing value by the number of current employees and then by the maximum possible number of employees. The supervisor who

ends up having the largest of both of these values is the supervisor whom the employee will attempt to get confirmation of employment from. Thereafter the supervisor will be placed into a "partially" hired supervisor set, and a message will be sent to the supervisor to confirm the employee's employment. The supervisor will then stay in this set until a confirm or disconfirm message is received or a timeout passes which will default to a disconfirm message. If a disconfirm message is received, the employee removes the supervisor from the "partially" hired set and continues seeking employment from other supervisors.

If a confirm message is received the supervisor will be placed into the employee's assignment subsystem, the only subsystem in an employee which can perform classifications for a hired supervisor. When an assignment message is received the record will be extracted and a predication will be performed, which will then cause a message to be sent back to the originating supervisor with the result. If a test is received at any time (an initial test should always occur to verify the employees knowledge) by a hired supervisor the employee will perform that test using its intelligence algorithm to generate predictions on the records being sent as the test. Once the test has been completed the test results will be sent back with the target classes the employee thought the records belonged to. The agent will then continue waiting for assignments, unless a failing grade is reported by the supervisor who initiated the test.

When a failing grade for a test is reported this will cause the employee to shift that supervisor from the testing to training groups. When this occurs the training subsystem will be notified of the supervisor and the employee will begin requesting at certain frequencies training data from that supervisor until the employee is satisfied they can be retested, whereby the training subsystem will then ask for a retest. This satisfaction algorithm is designed as an interface which returns a boolean value telling the training subsystem whether to submit a retesting job. The implementation currently used to return this boolean value allows for comparing the error rate of the neural network against a configurable value or using the number of training trials which have been processed by the employee for that

supervisor since the last retest. If the error rate is below the configurable value or the number of training trials is above a configurable value the implementation will then return true. Thereafter the employee will not request any more training units for that supervisor until a test is received, which the employee will attempt, and therefore may cause a shift of the employee from the training subsystem to the assignment subsystem for that supervisor depending upon the outcome of the test.

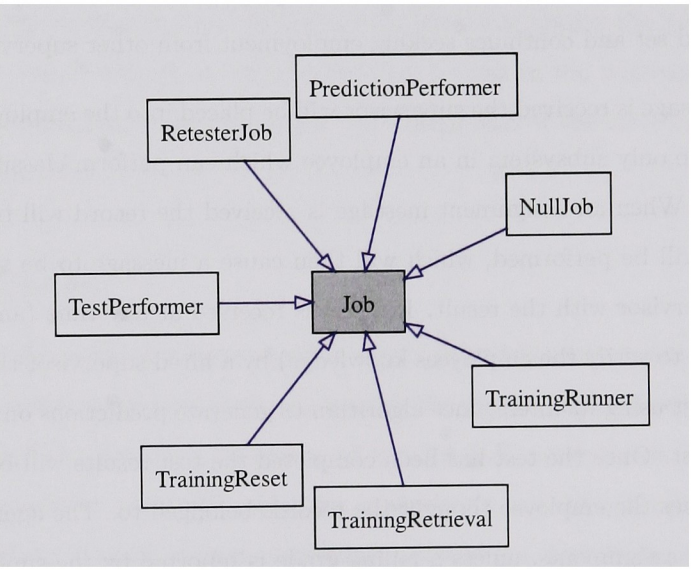


Figure 5.24: Job tasks

The tests, training, retesting, and assignment processes are all processed in a manner which ensures no two tasks are occurring at once. These tasks are implemented as job, which implement the Callable < T > interface provided in Java 1.5, with an abstract base class as shown in figure 5.24. The jobs once initialized are placed into a job queue for asynchronous activation in the order of their arrival by a separate job processing thread. Certain jobs also exist only to perform maintenance tasks such as ensuring the queue is actively running or to ensure that training retrieval jobs occur periodically for supervisors in the training subsystem.

5.2.6.3 Behaviours

In order to accommodate these employment actions, a method similar to the behaviours representation in the supervisor was chosen. This method allows all received messages to be channeled through a message processor who decides which employment subsystem should handle the messages content. The message processor itself observers the behaviours that do the actual receiving and receives notifications when messages arrive. This is implemented by following the observer pattern, where the message processor implements a message observation interface which the observable message receiving behaviours forward their messages to.

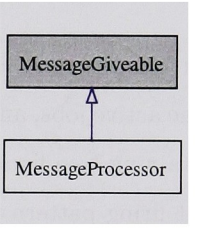


Figure 5.25: Message processor

In the case of this agent these behaviours are two behaviours which listen for messages with a performative of "confirm", "disconfirm" and "inform". When messages are received with this performative and the "employee" ontology by these two behaviours they are then forwarded to the message processor who will extract the content and then determine which employment subsystem the content should be given to. The current implementation listens for and processes messages corresponding to confirming and disconfirming employment, ending employment (firing), supervisor information, telling the employee they are assigning (passing grade causes this), telling the employee they are training (failing grade causes this), receiving training units and telling the employee to perform a test or prediction. If a supervisor sends these messages and they are active in an employment subsystem, the message processor will then either forward the message to that subsystem or activate a method call on that subsystem to tell it to perform a corresponding action.

Also implemented in this agent is an implementation of a liveness behaviour which will query the liveness manager to check the responsiveness of the supervisors the employee is employed from. This querying happens periodically and exists to ensure that the supervisors who the employee is connected are still "alive". If they are not responsive they will then be removed from the employment subsystem they exist in (testing or training), which therefore may cause the employee to resume searching for further employment. A similar process also occurs when a employee is told they are fired by a connected supervisor or when an employee ends the employment manually (i.e. when the employee shuts down).

5.2.6.4 G.U.I.

In order to have a meaningful G.U.I. for this agent an implementation was chosen which displays the neural network error rates, the active jobs, and the supervisors and which group they currently reside in, i.e. testing or training (see figure B.6 and figure B.7 in appendix B). In order to accomplish this, an event firing pattern with corresponding observers was followed. This allows the displayable units to fire events, which will be received by an observer which will then cause a display of the originating data in the interface.

In order to have the error rates displayed the G.U.I. follows the observable pattern and listens to the *DefaultNetProcessor* which fires off events when error rates change. This error rate is then shown on a JFreeChart graph which displays the error rate and the corresponding time that error rate occurred. Concurrently a table is also populated with the same information to allow for longer periods of time to be monitored. To enable the jobs to be monitored an event is fired when a job is added to a list of jobs, which the G.U.I. receives and displays the job in a JTable. Corresponding to this when the job is ran, and subsequently finished the G.U.I. also receives that event which causes the job display to remove the job from the same JTable instance.

To enable the supervisors to be shown in two separate JLists a similar procedure occurs, when they are moved between the two groups an event is fired which causes the supervisor

to be removed from the JList it currently exists in and shifted to the other JList component. This event is also sent when a supervisor sends a fired message or the employee sends a end employment message to the supervisor which then causes both JLists to check if the supervisor is in their list and remove them if they are.

5.2.7 Collectors

5.2.7.1 Proxy

The implementation of the proxy subsystem follows a rather straightforward process (seen in figure 5.26). The implementation first began by creating two separate storage units, one for the supervisors and one for the result receiver agents (which form the I/O subsystem). Thereafter using the F.I.P.A. subscription protocol[16], along with a subscription checking behaviour, the receivers request subscriptions to the collector while the collector performs the same request but instead on "discovered" supervisors. This request is sent out after ensuring the supervisor is "willing" to send messages to the collector (i.e. a subscription limit has not been reached in the supervisor). Subsequently, when a subscription is acknowledged and initiated following the F.I.P.A. protocol this then causes the receivers or supervisors to be placed in there corresponding storage unit.

Figure 5.26: Collector proxying process

Consequently when a message is received the message will have its originating supervisor identification removed (to be replaced with the collectors identification) and then the message will be handed off to the I/O subsystem to be sent to all units that are in the receiver agent storage unit. The J.A.D.E. implementation of these subscription protocols

makes this process very simple, as all that is needed in the implementation is to create a subscription "manager" for the receivers agents and a proxy behaviour to forward any received messages from the supervisor units to the subscription manager who then handles the broadcasting of the message (after subsequent modification) to all subscribed receiver units.

5.2.7.2 Behaviours

The behaviours which are implemented to aid in the proxying process are as follows, a subscription is maintained with the directory facilitator which notifies the collector when a supervisor has been initiated. These notifications then get forwarded to the proxy subsystem, which handles creating a behaviour which checks to see if supervisors are "willing" to allow any more subscriptions and if so a subscription initiation behaviour is added which performs the actual subscription request.

A similar behaviour also exists on the collector, but it responds to requests by receivers for subscriptions, and if the collector is "willing" (a configurable subscription limit has not been reached) to allow any more subscriptions a subscription confirmation is sent back (otherwise a disconfirm is sent). Thereafter these subscription behaviours follow the F.I.P.A. definition and the corresponding J.A.D.E. implementation of the subscription protocol.

Also implemented in this agent is an implementation of a liveness behaviour which will query the liveness manager to check the responsiveness of supervisors and the receivers the collector is subscribed to. This querying happens periodically and exists to ensure that the supervisors and receivers who the collector is connected are still "alive". If they are not responsive they will then be removed from the storage unit they exist in which therefore may cause the collector to allow further subscriptions from new receivers or supervisors. The same process is also used when a collector is told a subscription is canceled by corresponding supervisor or receiver. Similarly when a collector agent is deactivated, the subscriptions it is

currently maintaining are manually canceled by sending cancel messages to the supervisors and receivers the agent is maintaining subscriptions to.

5.2.7.3 G.U.I.

This agent has a corresponding G.U.I. which implements only a few necessary displays as there is not much information to potentially display (see figure B.8 and figure B.9 in appendix B). The information which is displayed is which receivers and supervisors the agent is connected to and proxying for, with the implementation of this information through two JList instances. In order to accomplish this "basic" information display it was decided that the storage unit classes which contain the supervisors and the receivers would implement an observable interface (following the observer pattern). Thereafter two observer instances would receive notifications when a supervisor or collector is added or removed, which would then cause corresponding G.U.I. events to be fired. The fired G.U.I. events would then be picked up by the G.U.I. event processor which would cause the corresponding JList to either remove or add identification information about the specific supervisor or receiver which caused the event to be fired.

5.2.8 Receivers

5.2.8.1 Results

In order to allow for meaningful results to be shown by this agent a selected set of classification methods for displaying results were chosen. These methods include methods to display a confusion matrix, false positive and false negative rates and pie charts to display the breakup of individual classifications as well as a table which displays the classification results received. In order to implement this many visualization methods each visualization implements an observer interface. This allows for the result visualizations to be notified when a classification is received and also allows for the classification reception behaviour (the observable) to not have to be concerned with how the visualizations are performed (as

can be seen in figure 5.27).

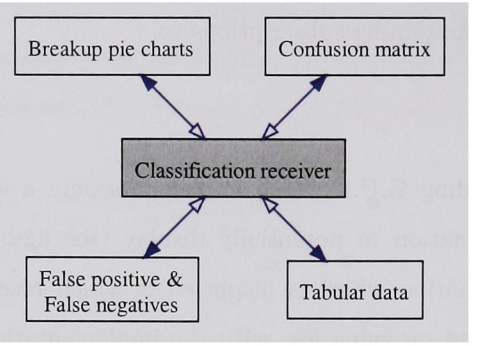


Figure 5.27: Visualizations

The confusion matrix, which is a matrix containing columns and rows, with the columns representing the predicted class while the rows representing the instances in an actual class allows for visual display of the classes which are commonly mislabeled. It also allows for a display of the classification error rate by calculating the number of classes not on the diagonal (the incorrect classifications) of the matrix and dividing this by the total number of classification results which have been received. The confusion matrix implementation in the G.U.I. is implemented as a JTable instance with a custom table model which dynamically changes its structure to accommodate for new classes as well modifying the corresponding matrix values when new classification results are received. The matrix itself is not stored directly in the table model, but instead a separate class maintains the matrix, which the table model then refers to.

A second table also displays the false positive and false negative rate if they can be calculated. The false positive rate which is the number of number of negative instances which were reported as positive along with the false negative rate which is the number of positive instances which were reported as negative are both displayed if the information necessary to compute these values exists. As it is necessary to know which class is "positive" to compute these values the properties file is consulted in the implementation to determine what this class should be. Thereafter when a classification result is received, and

a acknowledgment on the actual class has been performed with the originating datasource the false positive and false negative table will then be modified to reflect new rates if either rates have been modified.

The breakup pie charts are implemented as JFreeChart instances and they exist to visualize the percentages of each target class which has actually been classified as that class, with the other sections of the pie chart representing the classes it was mislabeled as. In the implementation a pie chart is created dynamically whenever a class which does not have a corresponding chart is received. Thereafter when a classification result is received for that class its values will be updated which will cause either a new pie section to be added or updated (if a misclassification is occurs) or will cause the target classes pie section to be updated if the classification is correct.

Finally a third table displays just the data received, with columns to display the originating datasource and record instance as well as the predicted and actual classes of the received classification. This allows for a fourth manner in which results can be analyzed if a tabular method is preferred, as this visualization is useful for exporting the classification results to programs such as Excel for further analysis.

5.2.8.2 Behaviours

In order to implement the visualizations described previously a set of behaviours are implemented which receive the classifications from collectors and then attempt to contact the originating datasource to determine the classifications instances actual target class. To receive results from the collectors a instance of the subscription initiator behaviour which J.A.D.E. implements is used to attempt to subscribe to located collector agents. In order to find these collectors another subscription initiator behaviour is maintained with the directory facilitator which will receive notifications when collector agents are initiated. Thereafter the implementation will attempt subscribe to subscribe to these located collectors if possible (i.e. the collector has not breached its subscription limit).

When a classification is received, an internal message is generated and sent with the received classification message. It will then be received by an internal "manager" behaviour who will extract the content of the message and initiate a asynchronous child behaviour which will attempt to contact the classifications originating datasource to determine the actual target class. Once this target class has been received (if the datasource still exists) the visualization units will then be notified of the classification and the corresponding actual target class, which allows them to perform whatever visualizations algorithms they implement.

5.2.8.3 G.U.I.

Since most of the G.U.I. has already been discussed (see figure B.10 and figure B.11 in appendix B), being that the visualization units compose most of the interface, there is one other G.U.I. components unique to this agent which is used to display the connected collectors. When a connected collector is acknowledge by the subscription behaviour, a notification is sent as an event to the G.U.I. which thereby causes the collector to be added to a JList instance in the interface. This allows for a visual display showing which collectors the result reception agent is receiving from. Similarly when a subscription is canceled an event is fired which causes the JList instance to remove the collector from the display unit.

5.2.9 Monitors

5.2.9.1 Monitoring

The monitoring system implemented by this agent follows a simple & straightforward process. The only actions it performs is to periodically initiate a liveness check for agents who have been located. When a response is received or the liveness check timeouts the monitoring system then notifies the G.U.I. with this information through a event being fired. Then after a period of time this agent will be checked again with this process continuing until the monitoring agent is deactivated.

5.2.9.2 Behaviours

In order to allow the monitoring system to operate a subscription is maintained with the directory facilitator which will notify the monitor when agents come online who have the liveness ontology listed as one of their implemented ontologies. Thereafter the liveness manager behaviour will receive periodic queries for these located agents. Upon a successful response or a timeout inherent to the liveness checking process a G.U.I. event will then be fired which will notify the interface of the outcome.

5.2.9.3 G.U.I.

In order to display the information related to the monitoring of these agents events are received which cause a JTable instance to be updated. This JTable instance and associated model show the agents identifier, currently residing container address, the last time they were checked for being responsive and if the result showed the were responsive or not responsive. This instance receives these events and adds or updates the corresponding agent information thereby allowing for visual feedback on the checks being performed. The generated G.U.I. which displays these components and responds to these events can be seen in figure B.12 and figure B.13 in appendix B.

This page intentionally left blank.

Chapter 6

Experiments

This chapter describes experiments developed and conducted to determine the accuracy, adaptability and fault tolerance of this thesis. In order to define these experiments, this chapter will first go over the method to be used in section 6.1 which will then be followed by the design of experiments in section 6.2 and the data-sets to be used in section 6.2.4. Thereafter section 6.3 will analyze the experimental results thereby allowing for analysis of this theses experimental methods and capabilities.

6.1 Method

In order to design experiments, the scientific method will be followed which applies an accepted series of steps to finding answers experimental questions. These steps will be followed by each experimental design in section 6.2 and corresponding result analysis in section 6.3. Along with this empirical method, the accepted approach for designing experiments shall be imposed. This ensures that there is an organized design to determine the relationship between experimental factors through comparison of experiments along with randomization and repetition of the experiment being performed to reduce the amount of variation in the outputs.

6.2 Design of experiments

6.2.1 Accuracy

An observation which was made as one of the founding principles of this theses design was that as more employee agents are added to a supervisors testing group, this has the potential to increase the quality of the classifications exported by that supervisor. The question therefore is to determine whether this increase in accuracy occurs, along with an associated decrease in false positives and false negatives, and to what limits will adding more employees have an effect on increasing accuracy.

The prediction is that as more employees are added, the supervisor will have more results, thereby by applying k-means the supervisor can then locate a centroid which will more accurately represent the actual target class. The hypothesis for why this should happen is that as more employees enter the testing group the centroid should be closer to the actual target class as the likelihood that all employees will misclassify the actual target class should be diminished.

In order to test this observation, prediction and hypothesis it was decided to design an experiment which has a single supervisor agent, a single datasource using a data-set composed of b target classes, and 2^n employees, where n initially starts at 0 and ends at k employees where $k > 0$. The other agents which would be included in this experiments set of agents is a single collector and result reception agent to allow for analysis of the classification accuracy. Thereafter the datasource will have its testing data-set classified by all of the n employees until that data-set has been depleted. Consequently, after three repeated iterations of this task for each value of n, with the data-set being randomized each iteration the data related to accuracy, false positives and false negatives (if applicable) can then be collected. Upon completion, up to $k = 3$, the data collected will then be analyzed to determine the trends in classification accuracy, false positives and false negatives as the employee amount is increased.

6.2.2 Adaptability

A second observation which was made as one of the founding principles of this thesis was that classification should occur with a greater level of adaptability when the capabilities of the implemented organizational structure are used to evaluate and adapt dynamically through the developed review and grading process. Therefore the prediction is that when new data-sets with unknown target types, with new classification data to what has already been processed, are added to a supervisor, the employees should be able to adapt to that data. The hypothesis why this should occur is the supervisors capabilities to evaluate the performance of employees and force retraining if an agent is not performing well.

In order to test this observation, prediction and hypothesis it was decided to design an experiment which will use a single supervisor, four employees and a collector agent and result receiver for classification result analysis. In order to have data to adapt to m datasources with h disjoint testing data-sets and corresponding validation sets (all data-sets will originate from a single larger data-set) will be activated. In order to ensure the data-sets are disjoint the originating data-set will first be split up into e disjoint data-sets, where e is the number of target classes. Thereafter for each repetition of this experiment, random subsets of these e data-sets will be chosen to create the h data-sets.

To start this experiment, a datasource will be initiated and used by the supervisor to initiate the employees and subsequently train and perform classifications on until that datasources data-set has been depleted. Thereafter this datasource will be disconnected and through an iterative initiation process, the rest of the datasources will be initiated and will have their data-sets classified until their data-sets have been depleted. In order to empirically observe the adaptation processes and the resulting effect on classification accuracy the previously described sequence will be repeated twice with two different disjoint sets for each larger data-set involved.

6.2.3 Fault tolerance

A third and final observation was made as one of the founding principles of this thesis. It was the observation that when you have multiple agents working on a classification, either directly or indirectly, and one or more of those agents fails, the operating quality and classification accuracy of the system should degrade gracefully. This observation therefore allows the designed architecture to have an operating quality which decreases proportional to the severity of failure, therefore incorporating one of the key components of fault tolerant systems.

Subsequently the prediction is that when n employees are initially established to a level where they can assume classification responsibilities and when each of those n employees then fails the effect on classification accuracy should decrease as some arbitrary function of the agents lost. The hypothesis for why the classification accuracy should degrade gracefully is the fact that the knowledge for performing classifications is not centralized at any one employee, but among all employees, thereby ensuring if an individual employee does fail the others can compensate.

In order to test this observation, prediction and hypothesis it was decided to design an experiment which will use a single supervisor, four employees, a single datasource (and associated validation and classification data-sets) and collector and result receiver for classification analysis. Thereafter the n employees will all be trained to a level where they all succeed in passing the supervisors tests, thereby allowing them to perform classifications. Subsequently a fraction of the testing data-set, totaling $\frac{m}{n}$ records, where m is the total records in that data-set will then be used to generate classifications, thereby forming a base classification level. Following this, an employee will be disconnected through agent shutdown, simulating a fault, and another, disjoint, $\frac{m}{n}$ records will be classified, with this process repeating until 0 employees are left. This process will then be repeated three times, therefore allowing for the hypothesized degradation of classification accuracy to be empirically observed.

6.2.4 Data-sets

In order to describe, initiate and subsequently run these experiments an intrusion data-set and a developed waveform generation application which can generate arbitrary waveform data-sets will be applied as the data inputs for all experiments.

The intrusion data-set defines network records and is itself derived from a larger data-set generated by M.I.T.'s Lincoln Labs in 1998. It has been used as the KDDCup'99 data-set and analyzed in multiple research materials[35, 36, 47, 48]. The data-set defines records for network instances, with each record representing twenty two intrusion incidents or normal network activity (the target class) and forty one derived network features which that network incident has generated. Prior to its use, the data-set had its numeric fields normalized into the range $[0.0, 1.0]$. Then the larger data-set was split into t smaller data-sets where each smaller data-set contains only one target type (resulting in a distribution shown in table 6.1). Finally to conclude the preprocessing the smaller t data-sets had their duplicate records removed and were randomized thereby ensuring an even distribution of records.

For the intrusion experiments which will be presented a data-set containing nine of the twenty two possible intrusion types (back, ipsweep, neptune, nmap, portsweep, satan, smurf, teardrop, warezclient) and normal records was generated. The nine were chosen due to the fact that all nine smaller data-sets can provide five hundred or greater records to the data-set being generated. Concurrently this also represents a sample of the original data-set which can be processed in a *reasonable* amount of time thereby expediting the experiment process.

For waveform identification, an application was created which can generate arbitrary data-sets based off of functions described sin and square waves. This G.U.I. application, which can be seen in figure B.21 and figure B.22 of appendix B, applies the following procedure for generating a data-set. For the two waveform types implemented, the application will prompt the user for properties related to the waveform, i.e. its frequency (fp), am-

target class	amount	processed amount
back	2,203	968
bufferoverflow	30	30
ftpwrite	8	8
guesspasswd	53	53
imap	12	12
ipsweep	12,481	3,723
land	21	19
loadmodule	9	9
multihop	7	7
neptune	1,072,017	559,203
nmap	2,316	1,554
normal	972,781	844,553
perl	3	3
phf	4	4
pod	264	206
portsweep	10,413	3,564
rootkit	10	10
satan	15,892	5,019
smurf	2,807,886	8,026
spy	2	2
teardrop	979	918
warezclient	1,020	893
warezmaster	20	20
total	**4,898,431**	**1,428,804**

Table 6.1: KDDCup'99 data-set distributions

plitude (a), noise multiplier (n), start value ($start$), increment value (iv) along with the number of records (ra) to generate and the number of fields per record (fr).

This process will initially start the function input c_{in} at $start$ and generate a function output c_{out} with that value as the functions input. After this output has been generated a randomly generated gaussian (normal) distributed noise value, with mean 0.0 and standard deviation 1.0, multiplied by n will be added to the output, thereby simulating a "noisy" waveform. Following this c_{in} will be incremented by iv and this process will repeat until $ra * fr$ values have been output. In order to generate the sin wave function using the parameters, at each iteration $a \sin(fp * c_{in}) + noise(n)$ is applied. For the square functions output, it first applies $\sin(fp * c_{in})$ and then tests the sign of this output, which if positive applies $1.0a + noise(n)$ while if negative applies $-1.0a + noise(n)$.

For the waveform experiments which will be presented a data-set containing the two waveforms with five different noise multipliers, $n = \{0.0, 0.1, 0.2, 0.3, 0.4\}$, was generated (segments of these can be seen in figures 6.1 and 6.2). For each of the ten variations three hundred records with twenty columns and one target type were generated, thereby representing 30,000 clean or noisy output values for each waveform. The configuration of each variation is as follows; the increment value for the sin and square waves is $\frac{\pi}{36}$ or 5° with a start value of 0.0 and an amplitude of 1.0 and frequency of 1.0.

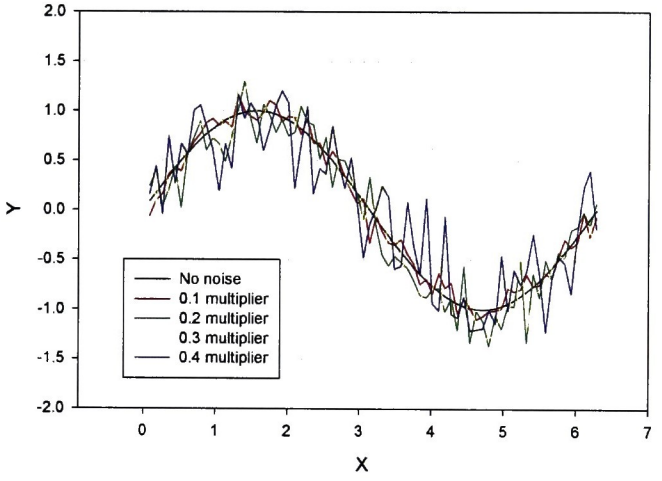


Figure 6.1: Sin waves

6.2.5 Agent configuration

In order to have a standardized configuration for all experiments, the following agent specific settings will be imposed. The datasource will use a data-set consisting of c testing records with t target types. Thereafter 33.33% of this will be used for a validation set, with the validation records selected being removed from the original c testing records, thereby avoiding reintroducing previously learned records. Finally to conclude, before a datasource

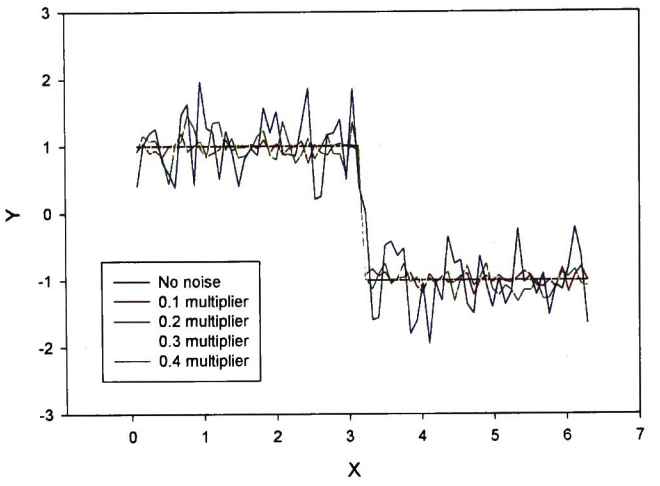


Figure 6.2: Square waves

goes online, it will randomize its testing and validation data-sets thereby ensuring an even distribution of target classes.

The supervisor will be configured to have a test passing grade of $\frac{75}{100}$ with associated test grade deviance being determined by the equation $\frac{1}{4t}$. For training and testing the supervisor will select a random 50% of its validation set and deliver this with or without its target types intact to the requesting employee to train or test on. An assignment passing grade of $100 - 100\frac{1}{2t}$ and a performance passing grade of $\frac{75}{100}$, with associated performance reviews occurring every 5 classification assignments. During this review process the past seven grades from the assignment, testing, and performance analysis processes will be used to generate a weighted average with tests being triple weighted, performance grades double and assignments retaining a weight of one. Retesting will also occur every 500 assignments for the accuracy and fault tolerance experiments and every 50 assignments for the adaptability and fault tolerance experiments; thereby allowing for the performance analysis and retesting configurations to dynamically evaluate and respond to employee

performance. Finally, during assigning processes the supervisor will request classification records through an iterative request process to ensure an order to the records classified.

As for the employees they will all be using a shared intelligence configuration to ensure the variation in intelligence units does not interfere with the experimental results. As is such, during each experiment a single neural network design will be used by each employee. For the experiments presented in this thesis the learning units will be M.L.P. networks with structures to match the data being classified. The structure of the network for the intrusion data-set experiments is as follows; a input layer consisting of 41 linear neurons outputs to a hidden layer with 25 sigmoid neurons over a full synapse which outputs to a second hidden layer with 20 sigmoid neurons over a full synapse which then outputs to a output layer with one sigmoid neuron over a full synapse. This network will use a learning rate of 0.7 with a momentum of 0.1. The structure of the neural network for waveform experiments will be as follows; a input layer consisting of 20 linear neurons outputs to a hidden layer with 25 tanh neurons over a full synapse which outputs to a second hidden layer with 20 tanh neurons over a full synapse which then outputs to a output layer with one sigmoid neuron over a final full synapse. This network will have a momentum of 0.1 and a different learning rate of 0.1; as the 0.7 learning rate was causing the networks to not converge. Each neural network unit will be configured to use 100 epochs being applied for each training job with the employee requesting a retest every two training jobs.

6.3 Results & analysis

The experiments provided a large number of variables for analysis and as each experiment was repeated k times this also introduces further data to be analyzed. In order to establish a meaningful result the data for the repetitions at each employee amount was averaged to form one set. Thereafter this averaged data representing the classification error rate (i.e. the misclassification rate), false positives, false negatives, neural network error rates and the grading data (i.e. the assigning, testing and performance grades) was analyzed to form

conclusive answers to the defined hypotheses. For those interested in the data itself and as it is to large for an appendix it can be found on the media accompanying this thesis.

6.3.1 Accuracy

In summary with regard to classification error rates for both data-sets and false positives and false negatives for the intrusion data-set; when the number of employee's is increased from one to eight employees, the error rate, false positives and false negatives rates does decrease over a single employee (the baseline). These improvements though tend to occur with a diminished return as the employee amount is increased, thereby suggesting an optimal number of employees for the tested data-sets. When compared to previous works the results validate the hypothesis that the application of a distributed adaptive framework applying social characteristics can aid in increasing the classification accuracy over non-distributed conventional models which apply a diverse range of well tested classification algorithms.

Intrusion data-set

For the intrusion data-set the average classification error rate for each of the three trials can be seen in figure 6.3. This graph shows the average classification error rate at each employee amount and tends towards an analysis that concludes that the classification abilities stabilize at three or four employees as further employees have minimal effect on the classification accuracy. This conclusion is reinforced by the average false positives and false negatives and associated rates which are shown in table 6.3.1 and the averaged confusion matrices for this experiment; available on the media accompanying this thesis.

The overriding factor for why this occurs is the fact that as the employee amount is increased, especially at the point where the number of employees reaches two, the variations in what the employees have learned starts to cause conflicts which results in differences on the agreed upon target class. An example is the baseline case, where the employee amount

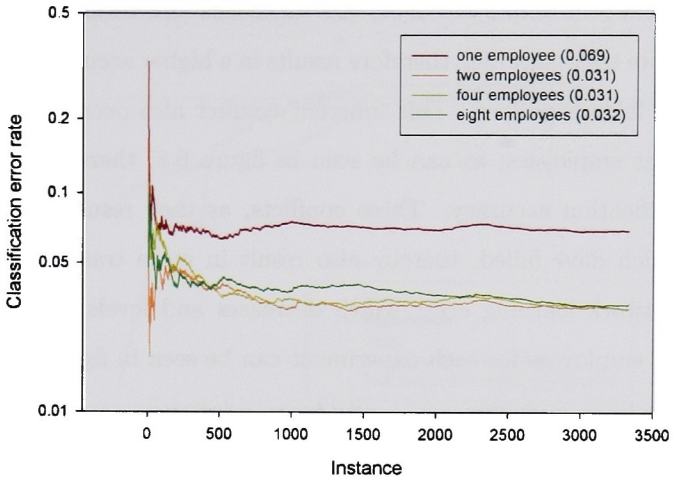


Figure 6.3: Classification error rates

Employees	False negatives	False positives	False negative rate	False positive rate
1	58	$31\frac{1}{3}$	17.31%	1.04%
2	$31\frac{1}{3}$	8	9.45%	0.27%
4	$29\frac{2}{3}$	$7\frac{2}{3}$	8.83%	0.26%
8	$32\frac{1}{3}$	7	10.02%	0.23%

Table 6.2: Average false positive and false negatives

is one, has a classification error rate almost double the other employee amounts. This is due to the fact that no conflicts in the target class ever occur since once the employee has reached the passing grade of the supervisor for testing and performing they will never fail an assignment, since there are no other employees providing input. This is reinforced by the data collected, which can be seen in figure 6.4, showing a quick rise in the test and performance grades with no change in the assignment grades throughout the duration of the experiment.

To contrast the single employee environment; the two, four and employee experiments achieved many more conflicts, which resulted in more training instances and more testing.

As can be seen in figure 6.5 and figure 6.6 which show the average grades over all evaluation instances of the two and four employees used, the variations and conflicts created among employees result in more training which therefore results in a higher accuracy and a decrease in false positives and false negatives. This inherent conflict also occurs similarly in the experiments with eight employees, as can be seen in figure 6.7, thereby resulting in its improvement in classification accuracy. These conflicts, as they result in more training for the employees which have failed, thereby also result in more training instances and therefore a neural network learning error which decreases and levels off. This training error averaged overall employees for each experiment can be seen in figure 6.8.

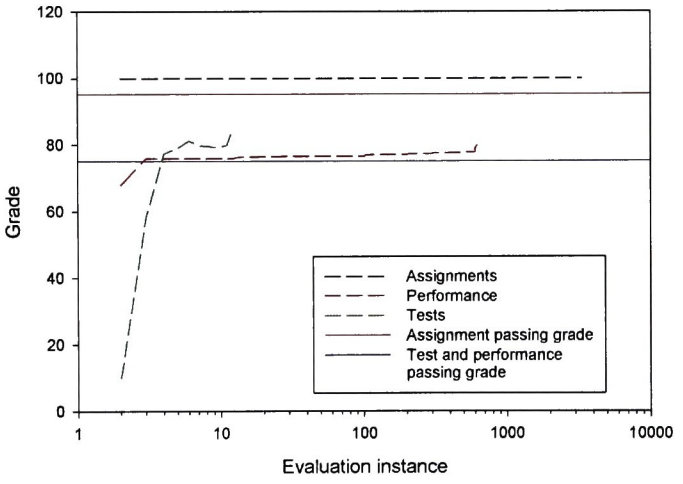


Figure 6.4: Single employee grades

As can be seen from the collected data and the graphs and confusion matrices referenced earlier the misclassification rate is directly correlated to the employee amount, up to four employees, therefore showing that this architecture results in a increased accuracy over the base case. At four employees the experimental data shown in figure 6.3 shows that as more employees are added, the current architecture does not show a statistically

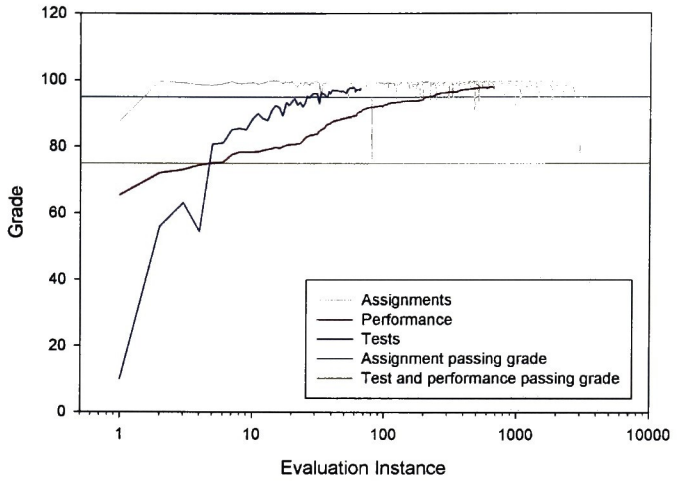


Figure 6.5: Two employee grades

significant improvement in the misclassification rate. The experimental data suggests that this occurs due to a saturation in responses, with four individuals providing enough variation in the target class during the assignment process to force individuals into retraining which eventually causes all four employees to begin to agree upon the target class of future records. This is also reinforced by the averaged neural network learning errors shown in figure 6.8 which shows that one employee is not sufficient to reach a stable learning error and two employees could do slightly better but above four employees the training errors and classification rates have stabilized.

Compared to previously performed research and experiments using the KDDCup'99 data-set the architecture has performed exceedingly well. Using the lowest average classification error achieved of 3.11% or a average recognition rate of 96.89% (shown in table 6.3 and in the corresponding confusion matrix) with a false negative rate of 8.83% and a false positive rate of 0.26% (shown in table 6.4) at four employees the architecture has been shown to be on the same level with previous research which typically only applies five of

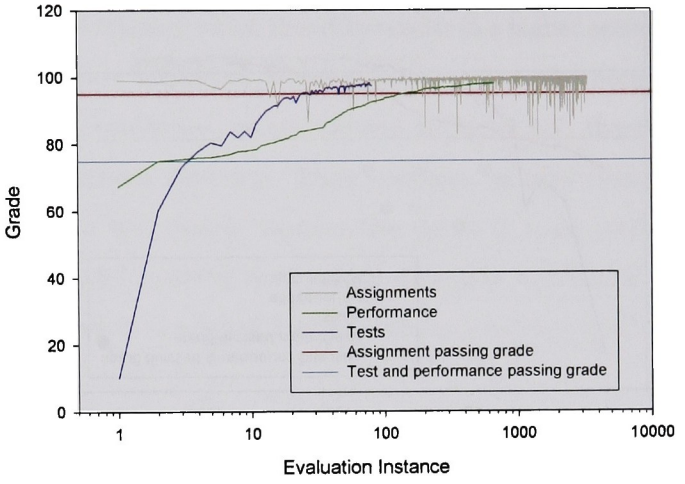


Figure 6.6: Four employee grades

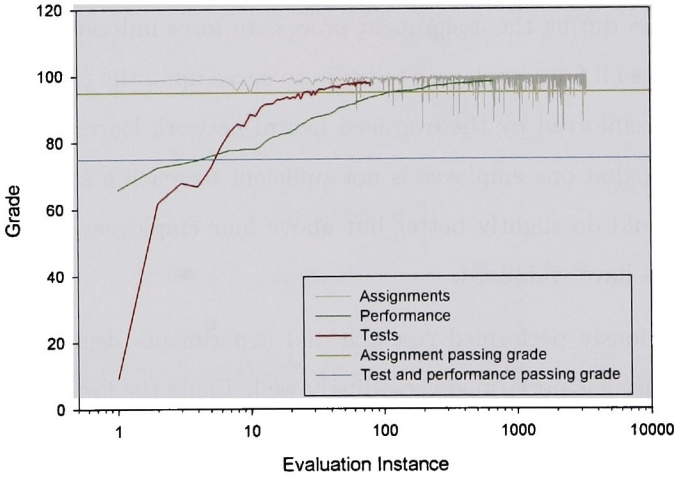


Figure 6.7: Eight employee grades

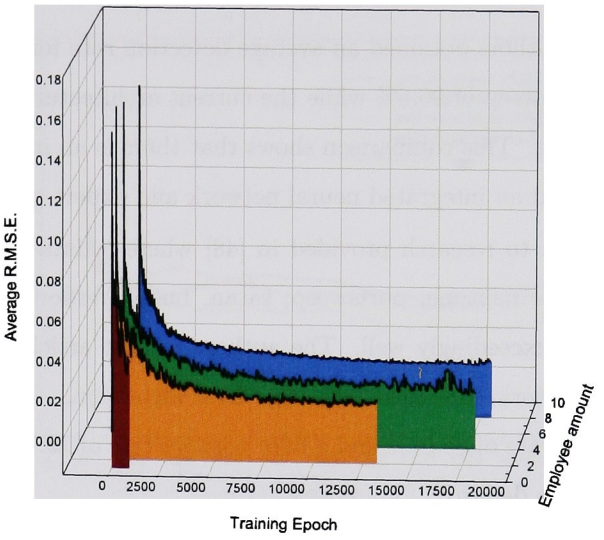


Figure 6.8: Neural network learning errors

Target class	Accuracy
normal	91.19%
back	98.19%
ipsweep	96.30%
teardrop	99.02%
smurf	98.81%
portsweep	98.52%
warezclient	97.98%
nmap	95.41%
satan	93.88%
neptune	99.59%
	96.89%

Table 6.3: Four employee average accuracy

False negative rate	False positive rate
8.83%	0.26%

Table 6.4: Four employee average false positive and false negatives

the ten target classes this experiment used.

In relation to [47] the authors obtained an average detection rate for the target classes of neptune, satan and portsweep of 96.6% while the current architecture obtained 97.33% for these three target classes. This comparison shows that there is an improvement over a previous work which applied an integrated neural network and expert system.

Similarly in comparison to research provided in [48] where a data-set was generated containing target classes for neptune, portsweep, satan, bufferoverflow and guesspasswd the architecture also does exceedingly well. The authors in that article applied a model consisting of a neural network combined with the C4.5 algorithm. They were able to obtain a average detection rate of 85.01% and a false positive rate of 19.7% thereby further validating this architectures design.

Finally in [43] the authors applied radial basis function neural networks and M.L.P. networks on a data-set containing equally distributed smurf, neptune, satan, ipsweep, back and normal target classes and associated records. They obtained a classification accuracy of 93.2% for the R.B.F. networks and a classification accuracy of 92.2% for the M.L.P. networks, both of which are comparable to the accuracy obtained using a single employee. This architecture on the other hand using four employees obtained a average classification accuracy of 96.33% for these target classes providing a noticeable improvement over a model which applies two classical neural networks.

Waveform data-set

For the waveform experiments, the classification error rates can be seen in figure 6.9 which shows that decreases in misclassifications do occur but they do not occur as profoundly as in the intrusion data-set experiments. As can be seen from the graph and the averaged confusion matrices for this experiment, available on the media accompanying this thesis, the averaged classification error rates over the three trials at each employee amount shows a decrease from 4.73% at one employee to a achieved minimum of 3.27% at four employees.

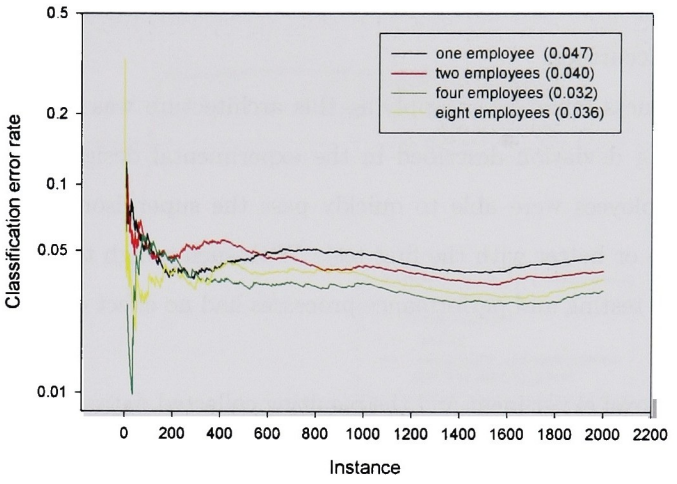


Figure 6.9: Waveform error rates

When compared to the baseline for this experiment the single employee, as no comparable previous research material exists, the conclusions show a pattern which is similar to the intrusion experiment. The single employee grades, shown in figure 6.10, as in the intrusion experiment show that the employee gets to a level where that employee passes the supervisors tests and performance blockades but never has any conflicts in assignments which will cause further training records to be requested. On the other hand, as with the intrusion data-set the two, four and eight employee average grades (seen in figures 6.11, 6.12, 6.13) show a much more vigourous assigning process whereby more conflicts do occur. These conflicts then result in more training data being processed which has the net effect of reducing the neural networks error, as can be seen in figure 6.14.

A conclusion which can be drawn from the experimental data is the correlation between misclassification rate and the number of employees. From the experimental data, the processes which occur indicate that four employees would be the optimal number of workers when this data-set is used. Having a higher value managed to increase the error rate, as

more conflicts occurred which then caused overfitting by having to many training iterations while having a lower value did not lead to the lowest achieved misclassification rate due to not enough conflicts occurring.

One notable weakness found when applying this architecture was the testing process which occurred testing deviation described in the experimental design. As the grading figures show, the employees were able to quickly pass the supervisor's tests, frequently achieving grades of 95 or better with the first test. With such a high testing grade, it can be concluded that the testing and performance processes had no effect on the classification results received.

In summary this novel experiment and the resulting collected data even with the weakness described previously further confirms the experimental hypothesis. As expected, when the employee amount is increased from one to eight employees the classification error rate decreases due to the grading processes which cause the employees in the experiment to be penalized for not obtaining classification results which are in the configured proximity with the centroid of the rest of the group.

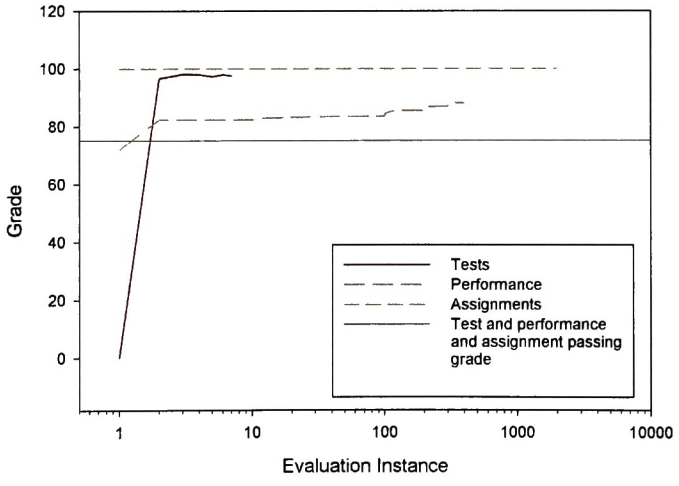


Figure 6.10: Single employee grades

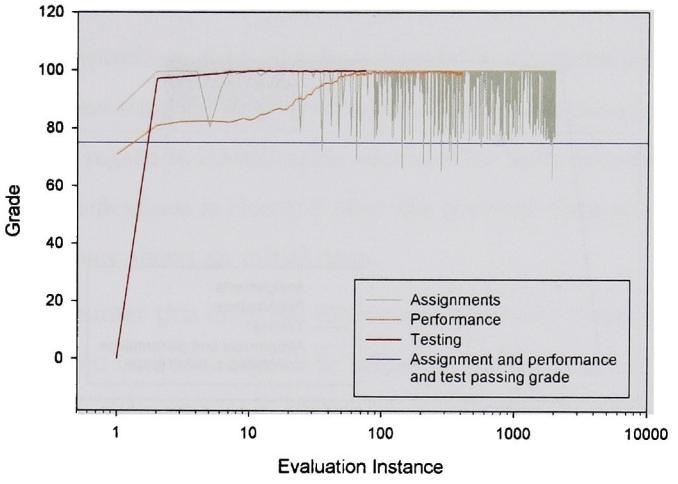


Figure 6.11: Two employee grades

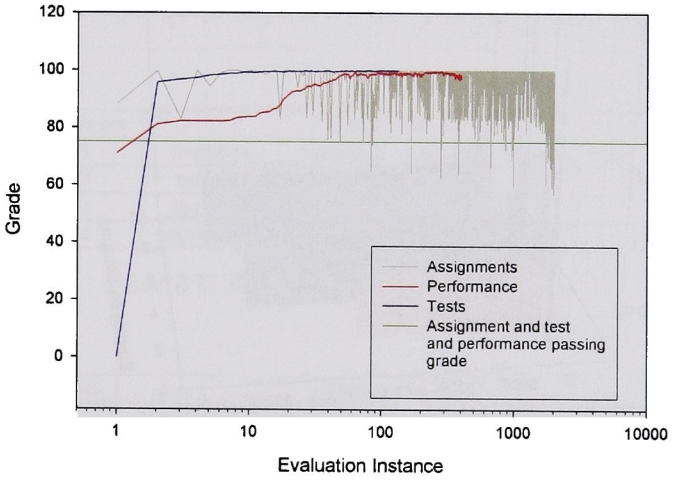


Figure 6.12: Four employee grades

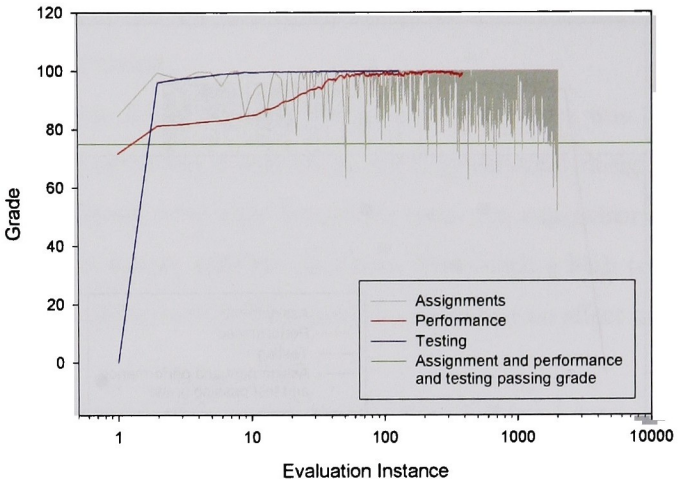


Figure 6.13: Eight employee grades

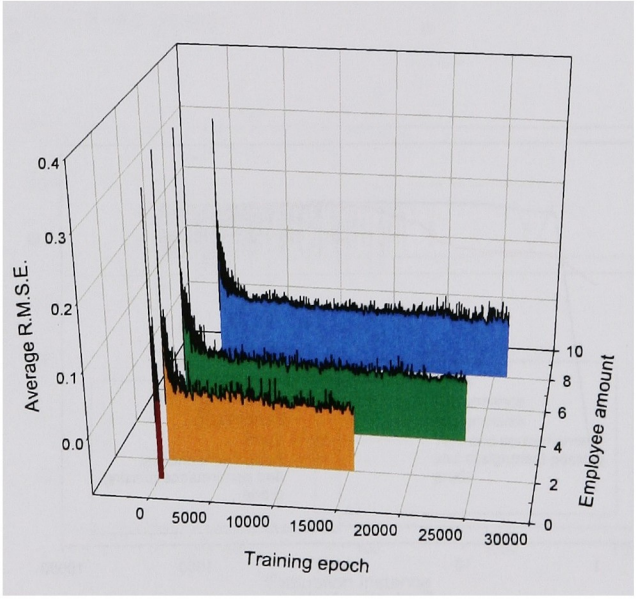


Figure 6.14: Neural network learning errors

6.3.2 Adaptability

For the intrusion and waveform data-sets the adaptability experiments proved to be very useful in further differentiating the developed design from conventional classification methods. In summary with regard to classification accuracy for both data-sets; when the data-set being used for classifications is changed after the previous data-set has been depleted the classification accuracy shows an initial drop.

In an attempt to counter this drop the supervisory processes composed of testing, performance and assignment analysis respond by initiating further training therefore allowing the classification accuracy to approach its former rate. This therefore supports the hypothesis that the architecture can autonomously adapt to new and previously unlearned data.

As described in the experimental design each target class is combined into a disjoint larger data-set with 0 or more other target classes (each row in the following tables) which is then applied by a datasource agent until it has been depleted. For the two intrusion trials the sets which were selected can be seen in table 6.5 and table 6.6 while the two waveform trials applied sets shown in table 6.7 and table 6.8.

Data-set	Target classes				
1	neptune	teardrop	nmap	smurf	back
2	normal	portsweep	satan		
3	ipsweep				
4	warezclient				

Table 6.5: First intrusion trials data-sets

Data-set	Target classes				
1	teardrop	neptune	normal	ipsweep	satan
2	portsweep	smurf	back	warezclient	nmap

Table 6.6: Second intrusion trials data-sets

Data-set	Target classes
1	square
2	sin

Table 6.7: First waveform trials data-sets

Data-set	Target classes
1	sin
2	square

Table 6.8: Second waveform trials data-sets

Intrusion data-set

The classification error rates achieved when running this experiment applying the intrusion data-sets can be seen in figure 6.15 and figure 6.16 (along with the related confusion matrices available on the media accompanying this thesis). These graphs show the global classification error rate and the effect on it when previously unlearned data is introduced. Simultaneously the graphs also show the local classification error rates which represents the post-computed local error rate of each data-set. This error rate was calculated by resetting the total number of records processed and the total incorrect records when each new data-set is initialized, thereby representing the classification error rate for the duration of that data-set only.

As can be seen from these two figures the hypothesis that the architecture can adapt to previously unlearned data, on a local or global error rate scale is verified as the spikes which occur in the figures, corresponding to a new data-set introduction. These spikes show that the transitions result in incorrect classifications as would be expected since the employees classify the new records as a type which they have been previously been trained on. If the architecture was solely applying a single conventional classification algorithm this would be the final result as the algorithm would not be readjusted and would continually misclassify the new data. Fortunately this architecture has procedures in place to counteract this stagnation, resulting in adaptation to the new data, through the supervisors aggregation and analysis processes.

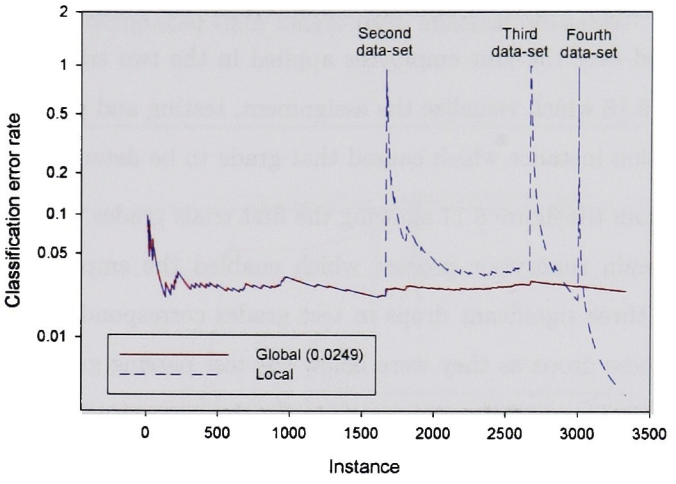


Figure 6.15: First trials classification error rates

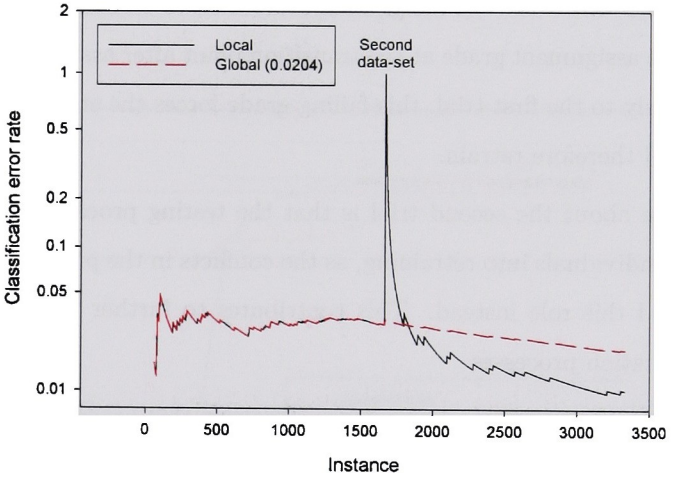


Figure 6.16: Second trials classification error rates

In order to understand how the adaptation and resulting error rate decrease was accomplished the supervisory processes and associated employees grades will be analyzed. These grades, averaged over the four employees applied in the two trials, can be seen in figure 6.17 and figure 6.18 which visualize the assignment, testing and performance grades relative to the evaluation instance which caused that grade to be determined.

As can be noted from the figure 6.17 showing the first trials grades, the testing process is shown to be the main supervisor process which enabled the employees to adapt to the new data, as the three significant drops in test grades correspond to the boundaries between data-sets. These drops as they were below the test passing grade thereby forced the employees to retrain by acquiring training data for the supervisor. As the supervisor would select the training data from the validation set generated by the newly arrived datasource the employee would therefore retrain and be forced to adapt.

In figure 6.18 on the second set of data-sets the assignment analysis processes were the main retraining process as variations in the classified result among the four employees caused those individuals who failed the assignment evaluation to be retrained. This can be seen as the drop in the assignment grade at the transition point after assignment evaluation instance 1665. Similarly to the first trial, this failing grade forces the employees to acquire new training data and therefore retrain.

Interesting to note about the second trial is that the testing process did not become involved to force the individuals into retraining, as the conflicts in the proposed assignment target class performed this role instead. This contributes to further validate the theses multi-dimension evaluation processes.

Unfortunately with relation to the performance analysis this process does not have a large enough effect to force retraining in this experiment beyond the initial failure which occurs for the first few performance analysis instances. Although the performance can be seen to change as the new data-sets are adapted to especially in figure 6.17, the change is never large enough to cause a failure. Therefore the experimental evidence suggests that

short term assignment analysis and testing processes are better predictors of classification ability when new data is injected than longer term analysis processes.

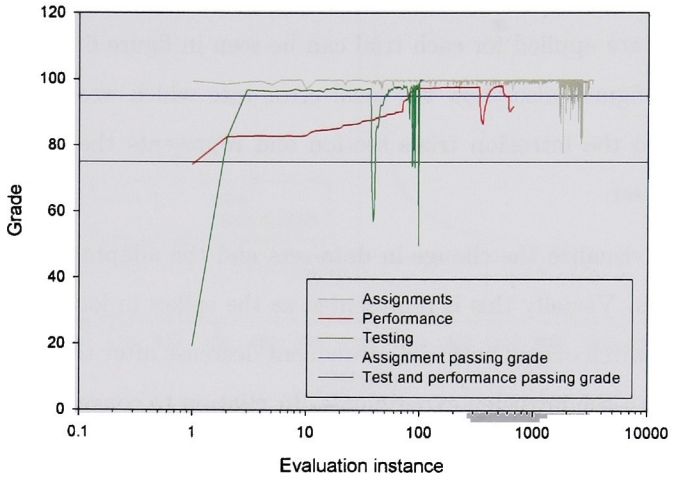


Figure 6.17: First trials supervisor grades

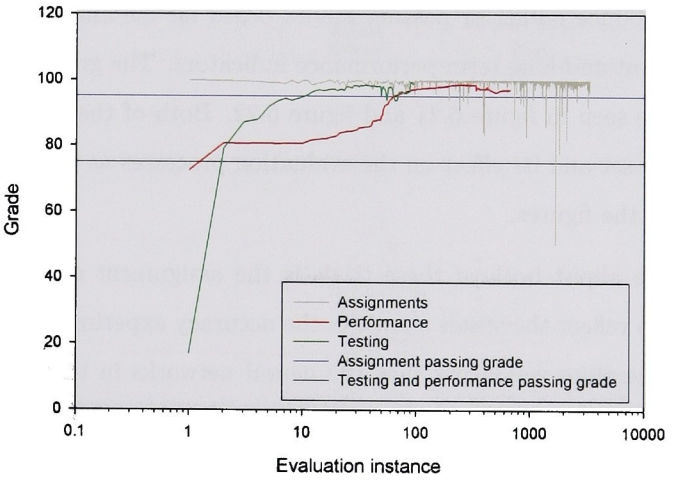


Figure 6.18: Second trials supervisor grades

Waveform data-set

The two experimental trials performed on the waveform data-sets shown previously provided results similar to the intrusion trials. The effect on the global classification error rate when binary sets are applied for each trial can be seen in figure 6.19 and figure 6.20. Simultaneously these figures also show the local error rate which was calculated by the procedure described in the intrusion trials section and represents the error rate for the duration of each data-set.

These two figures visualize the change in data-sets and the adaptation that occurs to readjust the employees. Visually this is represented as the spikes in local and global error rates as the data-set switch occurs with the subsequent decrease after the adaption occurs, thereby concurring with the intrusion experiments. In relation to conventional algorithms, if this architecture was solely applying a conventional classification algorithm there would be no further training and the error rate would continue to increase. Fortunately this theses supervisory evaluation processes autonomously react to the new data and force adaption.

As with the intrusion trials these evaluation processes originate through the supervisors capabilities to produce failing or passing grades based on the analysis of tests, each classification assignment and long term performance indicators. The grades resulting from these processes can be seen in figure 6.21 and figure 6.22. Both of these figures show the result of the new data-set and its effect on the evaluation processes as the drops in grades which are viewable in the figures.

Interesting to note about both of these trials is the assignment grades, which after instance 1000 being to reflect the states shown in the accuracy experiments graphs for the combined data-set; therefore validating that the neural networks in this experiment has the same difficulty when classifying the newly adapted data. Even with this difficulty the adaption (shown in the error graphs) which does occur further reinforces the hypothesis that the developed architecture can autonomously adapt to previously unlearned data.

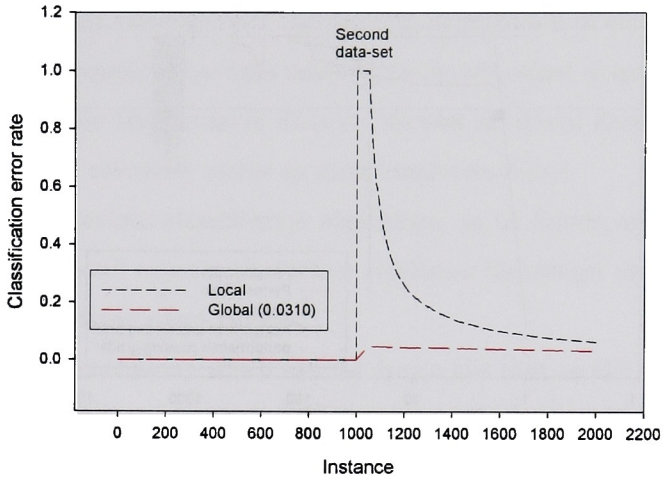


Figure 6.19: First trials classification error rates

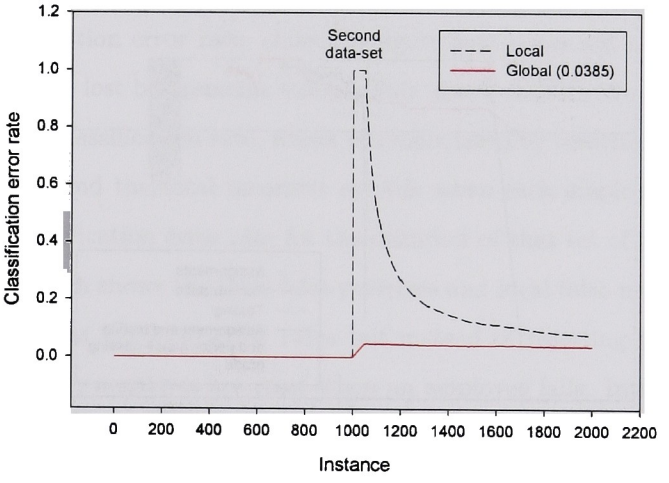


Figure 6.20: Second trials classification error rates

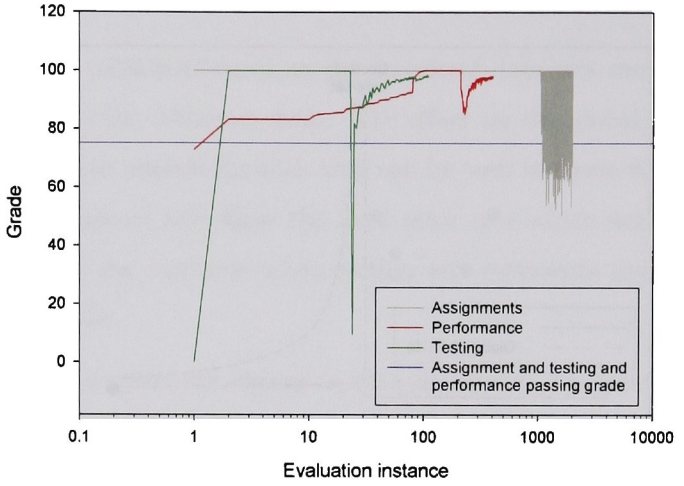


Figure 6.21: First trials supervisor grades

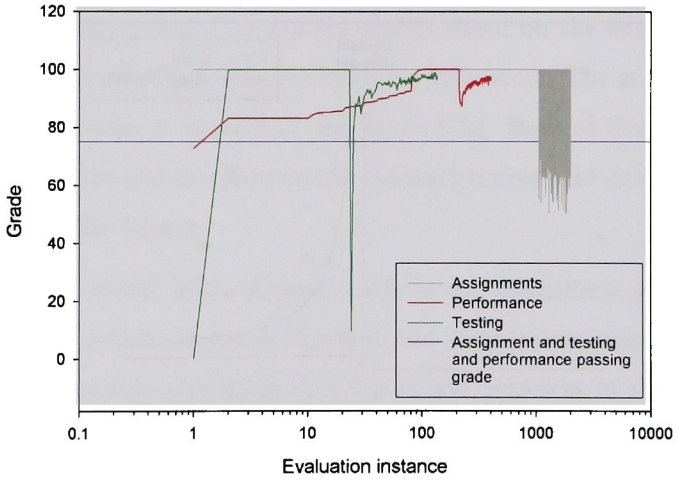


Figure 6.22: Second trials supervisor grades

6.3.3 Fault tolerance

With regard to the fault tolerance and the graceful degradation of classification accuracy as employees fail the experimental data showed the hypothesized drop in accuracy for the waveform data-set while the intrusion data-set showed an initial drop but unexpectedly the accuracy remained relatively stable as more employees failed.

With regards to previous classification algorithms, as no known architecture provides similar analysis, this novel experiment further reinforces the design this thesis introduces to accommodate classifying agent failures.

In summary, the conclusions which can be drawn are that as the four employees are reduced to one employee the conflicts which occur with the classifications being made before another employee fails aids in increasing the accuracy of the remaining employees. This increase then allows the remaining to better classify future records thereby causing minimal or no decrease in classification accuracy when the next employee fails.

Intrusion data-set

The intrusion data-set when applied in this experiment provided an unexpected result as the averaged classification error rate, shown in figure 6.23, does not show a degradation as more employees are lost but remains stable. This also is reinforced on the graph as the post-computed local classification rate, which was calculated by resetting the total number of records processed and the total incorrect records when each employee is lost, thereby representing the classification error rate for the duration of that set of employees only.

In figure 6.24, which shows the local false positives and local false negatives, calculated by the same process as the local error rates but instead of resetting the error rate, the false positives and false negatives are reset when an employee fails. Interestingly the false positives show this same unexpected result, where the false positives goes up initially as the employees learn the data-set and then goes down over the duration of the experiment. The false negatives though show the more typical pattern where graceful degradation does

occur as more employees are lost.

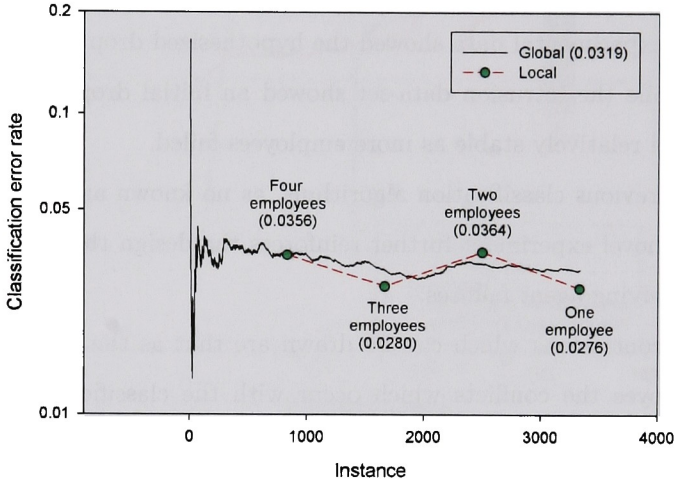


Figure 6.23: Classification error rates

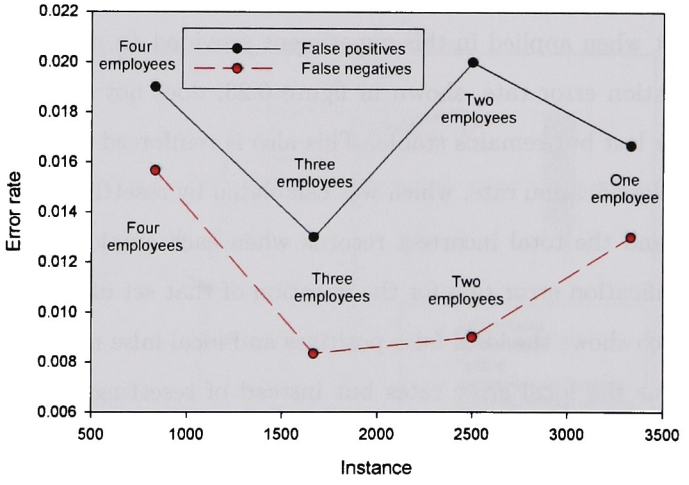


Figure 6.24: Local false positives and false negatives

In order to explain why this happens, the supervisor grading processes, visible in figure 6.25 must be analyzed. The assignment grades are the major factor in why the classification accuracy and associated error rates are maintained. The grades show the large number of conflicts which are occurring up to two employees (the dotted vertical lines each represent an employee being lost). These conflicts cause more training to occur and thus improve the accuracy of each employee so that as the next employee is lost the remaining employees perform even better and are able to compensate for the lost employee.

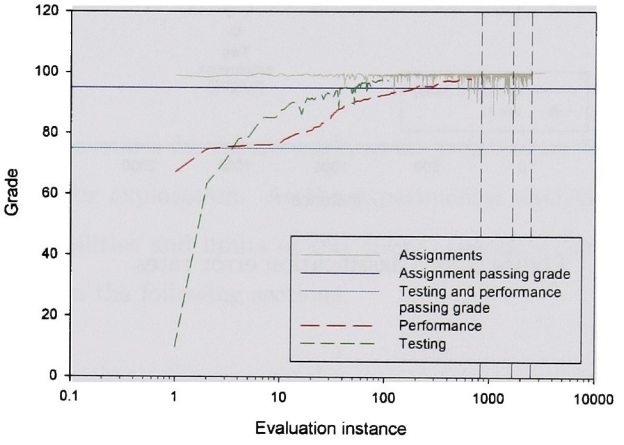


Figure 6.25: Employee grades

Waveform data-set

Unlike the intrusion data-set, the waveform data-set provides much more atypical result with regard to the overall classification error rates and shares the same explanation for why this occurs as the intrusion data-set. As can be seen in figure 6.26, the error rate initially at three and four employees is stable, as the employees have benefited from the retraining created by the assignments, testing and performance grades (seen in figure 6.27) while at the end of the experiment the single employee performing the classification is not able to

obtain the same level of accuracy since no further retraining occurs, thereby explaining the degradation which occurs.

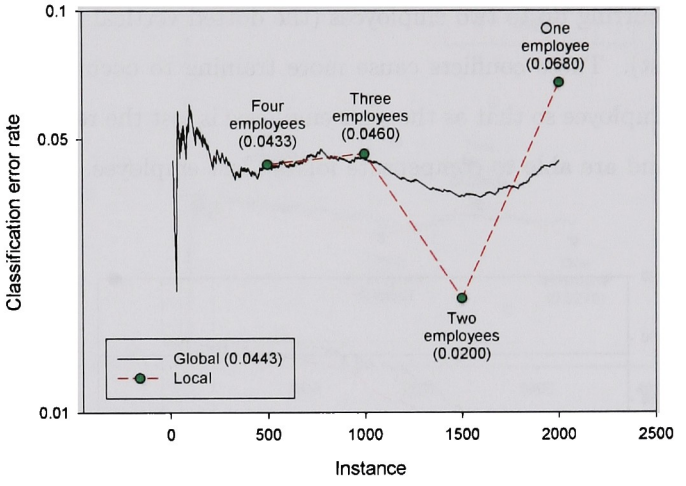


Figure 6.26: Classification error rates

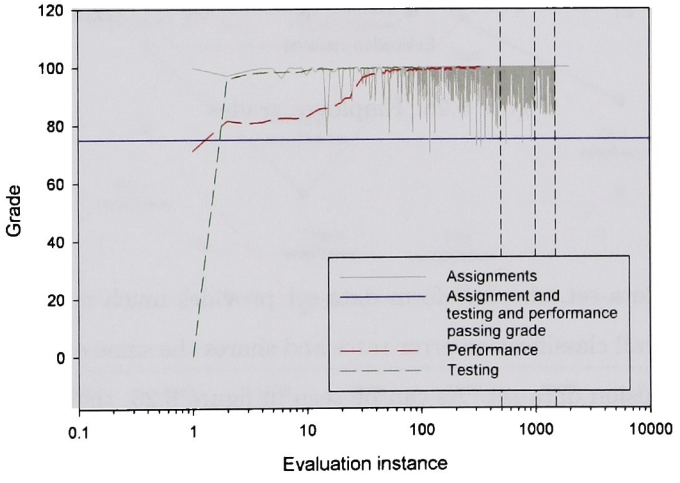


Figure 6.27: Employee grades

Chapter 7

Future work

This thesis opens many avenues for future work, as an architecture like this one opens up many new possibilities for exploration. As the experimental analysis has only scratched the surface of the capabilities and limits of this thesis, therefore various ideas for future work will be addressed in the following sections.

7.1 Agents

One avenue which could be explored is to generate employee agents which do not just apply neural networks as their underlying classification mechanisms. This could involve applying algorithms such as ID5R or other iterative classification algorithms and comparing their performance/accuracy. This could therefore open up more of the potential of this framework as different combinations of algorithms could be exploited. A second avenue which could be explored is to test more agents, as J.A.D.E. allows for unlimited number of running agents. Due to thesis goals, resource and time constraints this thesis was not able to test a large amount of agents (potentially with varying sets of neural networks). It would therefore be interesting to determine the performance of a large number of agents and the effect on classification accuracy. In order to accomplish this goal it would be necessary

to thoroughly examine previous papers relating to J.A.D.E. network performance such as [60]. Along this line a method whereby agents transfer their learning back-ends from good to poorly performing agents could be implemented. The practicality of this could enable agents whom have learning back-ends which are doing well relative to the group could transfer the back-ends to others who are performing poorly, thereby instantaneously increasing the receivers accuracy. This process would preferably be implemented with some type of genetic algorithm process to ensure the receiving agents do not receive exact duplicates of the senders back-end, thereby avoiding getting stuck in local minima. Another area which could be explored is to take advantage of the mobility capabilities built into J.A.D.E.. Incorporating these capabilities would allow for agents to move from container to container, thereby further encouraging greater diversity.

7.2 Data

As for the data involved in this framework, the algorithms used to perform pre-processing could be investigated in greater depth. Another avenue which most definitely needs improvement is the iterative process used for classification, whereby individual records are sent and the results analyzed. Future work could improve this process by handle blocks of records to improve the overall speed of classification. Future work could also analyze how streaming data, which would be received in blocks or single instances, could be implemented to allow for streaming of time-series data (or similar streaming data) to be analyzed. Also future work could allow for non-numeric types to be used throughout the framework, instead of the current implementation which forces these non-numeric values to be converted to numerical values. Along these lines a future set of modified agents could be implemented allowing for n data formats to be running on those agents at once, thereby requiring only one group of agents instead of the current implementation which requires n disjoint groups.

7.3 Algorithms

As for the employment algorithms used by both supervisor and employee agents it would be interesting to attempt different methods of rating the agents. This would be rather simple as the algorithms for these agents are abstracted to a high enough level where they can be easily swapped in and out. Therefore some type of comparison could be implemented comparing the analysis and overall effect of the different rating methods on the entire group. Along these lines it would be an interesting area of future work to enable supervisors or employees to communicate among themselves (possibly through blackboards or some type of similar "meeting" system) to aid in the classification processes. It would likely be applicable in this area of future work to consult organizational theory or other similar resources to determine what those resources would recommend for algorithms or methods.

This page intentionally left blank.

Chapter 8

Conclusions & implications

This thesis presents a architecture for data classification applying mutually adaptive distributed agents which follow a hierarchy with similarities to the neocortex. From the ground up this multi-agent distributed architecture was designed to allow for arbitrary data to be autonomously classified using a multitude of potential classification algorithms. Through a novel design which applies agents with a range of organizational duties the experiments have proven that the developed architecture is on the same level or better than previous work with relation to classification accuracy. Furthermore this architecture is shown to provide significant benefits in areas in which previously unlearned data must be adapted to or in areas where faults or failures are common. Simultaneously this thesis provides further validation that distributed social entities can enable benefits which are inherently not available in conventional methods.

As this thesis presents experimental evidence for improved accuracy, adaptability and fault tolerance the real world applications which could apply this architecture's design span a diverse range of different application areas. Potential real world applications include commercial and research areas where classification is commonly applied including intrusion and virus detection, insurance and health data mining, web patterns mining, email analysis and letter and voice classification to name a few. With little or no modification to the

underlying source code composing this thesis each of these areas could take advantage of the benefits the experimental evidence has shown to exist.

In conclusion, after working on this thesis it has become apparent that the research area which this thesis develops in has an unlimited potential for future work. Even though the area this thesis developed in is only a small part of artificial intelligence the potential shown in this thesis has the capability to open up new areas of research that simulate other areas of the neocortex. It is speculated that once this has been completed, with mutually adaptive distributed agents for areas such as image, voice recognition, emotions and memory; the agents then could be connected together to form a conglomerate model that approximates the full capabilities of the neocortex. The implications of such a model, which would inherently need to efficiently manage and direct the varying sub-models, are hard to even comprehend as nearly all areas of artificial intelligence could be affected.

"Genius is 1% inspiration and 99% perspiration."

Thomas Edison

1847-1931

Appendix A

User guide

In order to allow for potential users to use this framework, the following chapter will present a guide as to the procedures to use the created architecture. In order to accomplish the goal of presenting a useful user guide the chapter will be broken down into the following tasks each with a corresponding section on how to accomplish that task. Interfaces for the mentioned applications can be found in appendix B.

[Prerequisites tasks are in brackets]

(Task sections are in parenthesis)

1. Compiling the source [∅](A.1)
2. Configuring global parameters [A.1](A.2)
3. Launching a main container [A.1](A.3)
4. Launching agents [A.1, A.2, A.3](A.4)
5. Activating agents [A.4](A.5)
6. Examining agents [A.5](A.6)
7. Additional applications [A.1](A.7)
8. Further documentation [∅](A.8)

A.1 Compiling the source

In order to use the architecture, the source code for this architecture must be compiled to Java bytecode. To accomplish this task you must locate the media accompanying this thesis and replicate the source ("src" folder), libraries ("lib" folder) and the "build.xml" file required to compile the architecture. Also ensure that you have a Java 1.5.0 or later sdk and ant* installed as these are necessary to compile the source code in the most straightforward fashion. Once these have been installed and the appropriate files copied, you can then type the command "ant" on the command-line in the directory where you have copied the required files (the output should look like figure B.1).

A.2 Configuring global parameters

In order to accomplish this task, an additional file must be copied from this media accompanying this thesis. The configuration file that needs to be copied, or generated (ensure you follow the XML schema available from Sun†) in some manner is the file from the configuration ("config") directory labeled "properties.xml" to the location where you have placed the source and "build.xml" files. Ensure that you also place the file into a folder labeled "config" or modify the "build.xml" file appropriately to tell ant where this file exists. This file contains a documented set of properties which can be modified to alter the behaviour of the agents running in the system. A second way to modify these properties, is to launch the starter application and configure them in this manner (the file will not be saved). In order to do this approach, type "ant Starter" on the command line where the thesis code and associated files have been copied to (ensure you have compiled the source first). Thereafter a dialog will be shown where the properties of the "properties.xml" file can be modified (as can be seen in figure B.19).

*`http://ant.apache.org/`
†`http://java.sun.com/dtd/properties.dtd`

A.3 Launching a main container

In order to achieve this task it must be first told that only one main container needs to exist for agents to connect to. Therefore this task only needs to be done once, although multiple main containers can exist if it is desired. To start the main container, run the starter application, using the ant command "ant Starter" and go to the actions menu. Then select the "start a main container" option which will prompt a dialog (as shown in figure B.18). Then select the port you wish to run the main container on and whether to display a J.A.D.E. G.U.I. environment along with an option to disable or enable mobility on the container. Once you press "ok", the main container will be generated on the localhost if it does not already exist on that port and it will then allow agents to connect to it through your network address.

A.4 Launching agents

After you have completed the necessary prerequisites you then need to copy or generate (if you generating ensure you follow the XML schema) another file, this one which the starter application will use to populate its knowledge of the agents which can be started. This file should be located on the media accompanying this thesis in the configuration folder ("config") with the label of "agents.xml." Ensure that you place this file in a folder labeled "config" or modify the "build.xml" file appropriately so that ant knows where to look for this file. Also necessary is the folder labeled "schema" which contains the XML schema for the various XML files you shall be using (except the properties file schema provided by Sun). Once this has completed you can then select an agent to launch and provide it a name, a container name and the host and port where the main container exists. Thereafter go to the action menu once you have selected an agent and filled in the necessary properties and select "start agent." Once this has been completed the agent should then start up if there are no problems, otherwise check the console output.

A.5 Activating agents

A.5.1 Datasources

To configure and activate the datasource agent, you must have first launched this agent type via the starter application as well as loaded a two critical file for this agent which tell it what fields will be used in any accompanying data and how to convert nominal fields to numeric values. These files are specified in the "properties.xml" file under they key values of "fieldmapping" and "mapping". These must be modified accordingly (following the corresponding XML schema) to ensure any opened data-set files will load. After this is done you should be able to launch the datasource agent (which should result in a G.U.I. application as shown in figure B.2). Once this has occurred, you may then load or insert data, which has fields and nominal values corresponding to the files you just specified through the "file". Upon loading, the loaded data should shown in the left side of the main window with the same data appearing in the right side of the window but with nominal to numeric replacements being performed. If autovalidation was selected in the control menu, then a validation set should also have been generated. Otherwise you must right click on the records you wish to use in the validation set and create this validation set manually. Thereafter once these files have been loaded, you can then start the agent by going to the control menu and selecting the "start" action. Consequently the started agent will then be online and will perform the associated duties described in this thesis until the application is closed or told to stop through the same control menu.

A.5.2 Supervisors

To configure and activate the supervisor class of agents, you must have first launched the desired supervisor agents via the starter application (which should result in a G.U.I. application as shown in figure B.4). Once this has been achieved you then must go to the "control" menu and select "start". Subsequently the agent will be online and will perform

all of the duties as specified in this thesis until the application is closed or told to stop through the same control menu.

A.5.3 Employees

To configure and activate the employee class of agents, you must have first launched the desired supervisor agents via the starter application (which should result in a G.U.I. application as shown in figure B.6). Once this has been achieved you then must go to the "file" menu and load a corresponding neural network. Then you should go to the "control" menu and select "start". Subsequently the agent will be online and will perform all of the duties as specified in this thesis until the application is closed or told to stop through the same control menu.

A.5.4 Collectors

To configure and activate the collector class of agents, you must have first launched the desired supervisor agents via the starter application (which should result in a G.U.I. application as shown in figure B.8). Once this has been achieved you then must go to the "control" menu and select "start". Subsequently the agent will be online and will perform all of the duties as specified in this thesis until the application is closed or told to stop through the same control menu.

A.5.5 Receivers

To configure and activate the receiver/reception class of agents, you must have first launched the desired supervisor agents via the starter application (which should result in a G.U.I. application as shown in figure B.10). Once this has been achieved you then must go to the "control" menu and select "start". Subsequently the agent will be online and will perform all of the duties as specified in this thesis until the application is closed or told to stop through the same control menu.

A.5.6 Monitors

To configure and activate the monitor class of agents, you must have first launched the desired supervisor agents via the starter application (which should result in a G.U.I. application as shown in figure B.12). Once this has been achieved you then must go to the "control" menu and select "start". Subsequently the agent will be online and will perform all of the duties as specified in this thesis until the application is closed or told to stop through the same control menu.

A.6 Examining agents

In order to gain useful knowledge about what the agent and associated applications are doing at a given time a certain set of G.U.I. resources were implemented. Each agent contains an "about" menu which will activate a dialog which displays useful information about the agent, its behaviours and properties and its current runtime environment (see figure B.14). Along these lines each agent also has a section of its G.U.I. dedicated to logging, the messages sent and received and charts showing bandwidth information which displays the amount of network usage the transmission of these messages require (see figure B.15). Additional certain agents display runtime activities directly in their main G.U.I. window, such as with the supervisor and employee agents which display dynamically updating information relating to employment processes.

A.7 Additional applications

A.7.1 Data processor

In order to use the data processor, if you wish to perform basic tasks on data-sets the following procedure must be followed. First you must have compiled the source files as a prerequisite and then you can launch the data processor with the command "ant Comman-

der". Thereafter a screen should appear allowing you to perform various actions data-sets (which should resemble figure B.16). To use this application all that is required is that you select a data-set zip file, an action to occur and an output directory. If any probelms occur, whether in file reading, parsing, or action processing the G.U.I. or the console in the G.U.I. will notify you that errors have occured.

A.7.2 Neural network repair

To use the neural network repair utility the following processes must be followed. First ensure that you have completed this tasks prerequisites and then type the command "ant Utility" which should then result in a G.U.I. application as shown in figure B.20. After this has been accomplished you can then load neural networks created by the Joone editor or Joone neural networks which have been hand-serialized and specify the necessary input layer, output layer and network names as desired. If these layers are not found or other errors occur when you attempt to repair the network you will be visually notified and therefore can take appropriate actions to resolve the errors.

A.7.3 Waveform generator

To use the waveform data-set generator the following processes must be followed. First ensure that you have completed this tasks prerequisites and then type the command "ant Waveform" which should then result in a G.U.I. application as shown in figure B.21. After this has been accomplished you can then select waveform types to generate from the "control" menu. Once one of these is selected a waveform configuration dialog will be displayed, which can be seen in figure B.22 allowing the user to configure that waveforms output. Thereafter the generated input and output data-sets will be displayed in the G.U.I. and can be saved by going to the "file" menu and selecting save, thereby outputting a C.S.V. file.

A.8 Further documentation

For source code documentation, on the media accompanying this thesis there should be a folder labeled "documentation." This folder will have in it two different version of the documentation for the source code in this thesis. One folder will be labeled "javadoc" and will contain the documentation in a html format familiar to Java users. Another folder will be labeled "doxygen" and will contain the documentation in two additional formats (with the same content) produced by the popular documentation tool doxygen[‡]. One of these formats will be html while the other will be a compiled help file (chm) which is native to Windows platforms and useful for reading the documentation in a "book" like format.

[‡]`http://www.doxygen.org/`

Appendix B

Interfaces

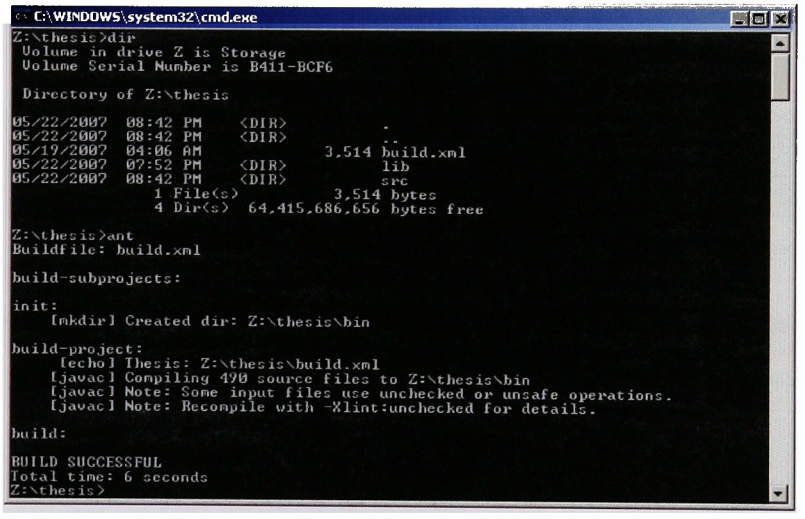


Figure B.1: Ant build

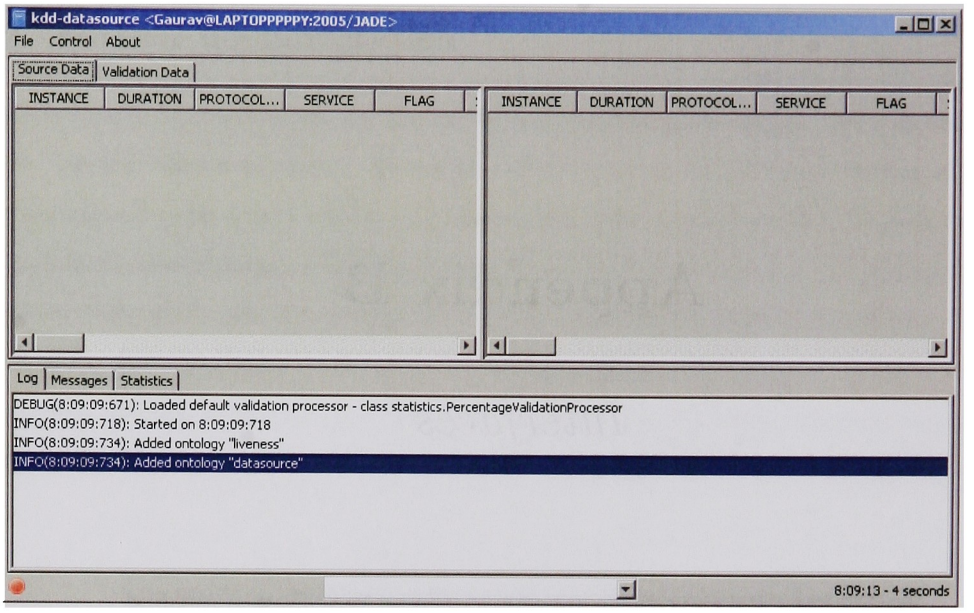


Figure B.2: Datasource inactive

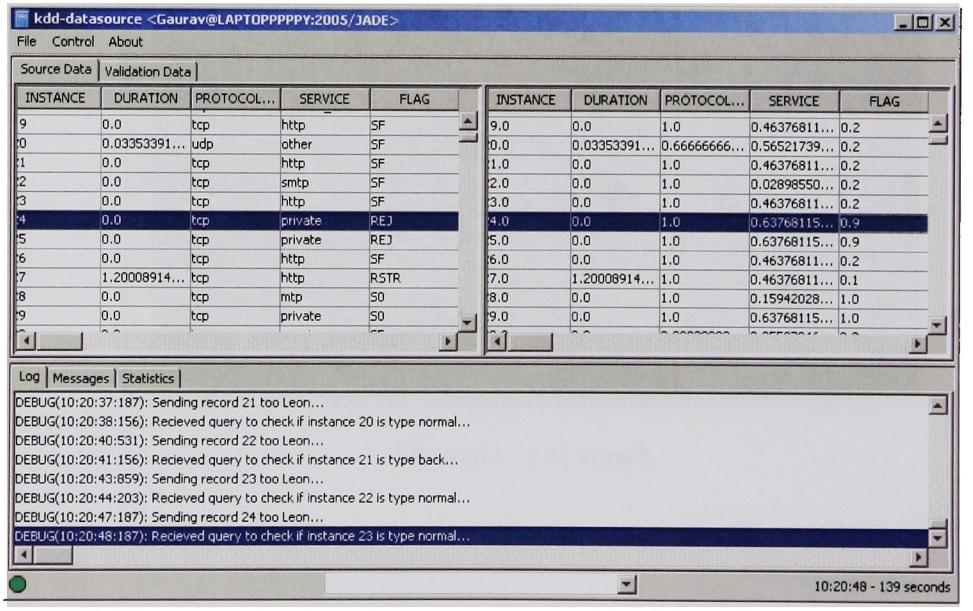


Figure B.3: Datasource active

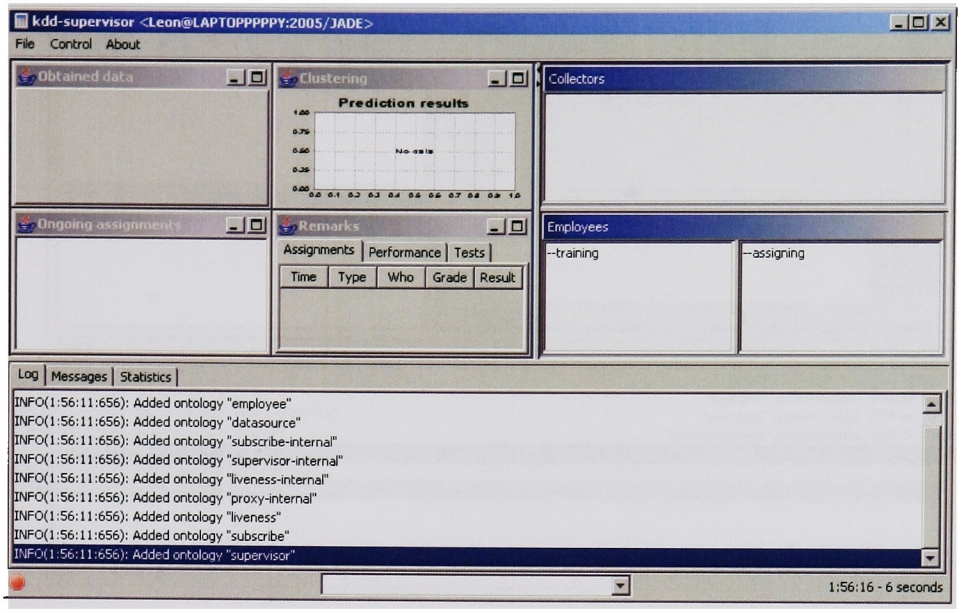


Figure B.4: Supervisor inactive

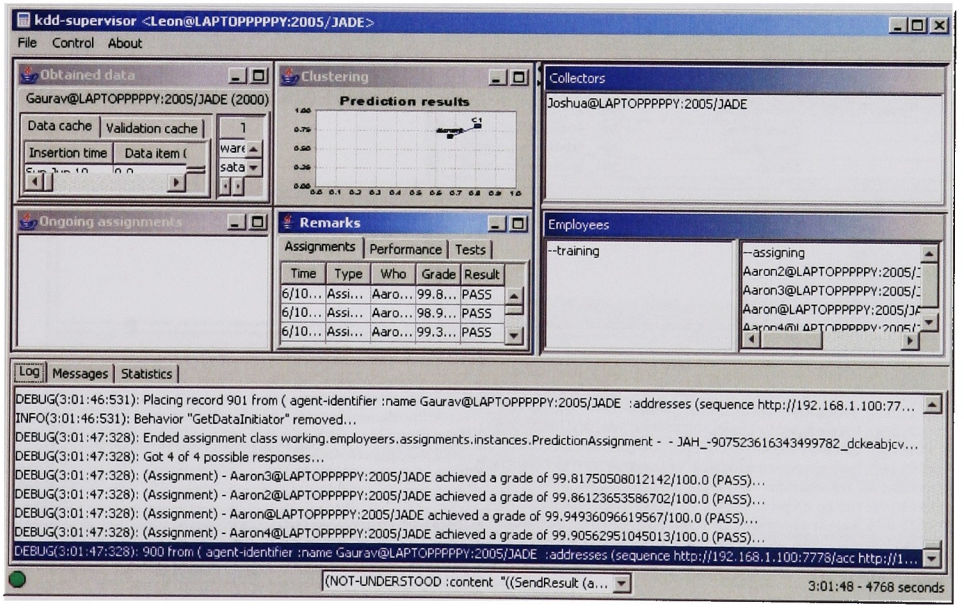


Figure B.5: Supervisor active

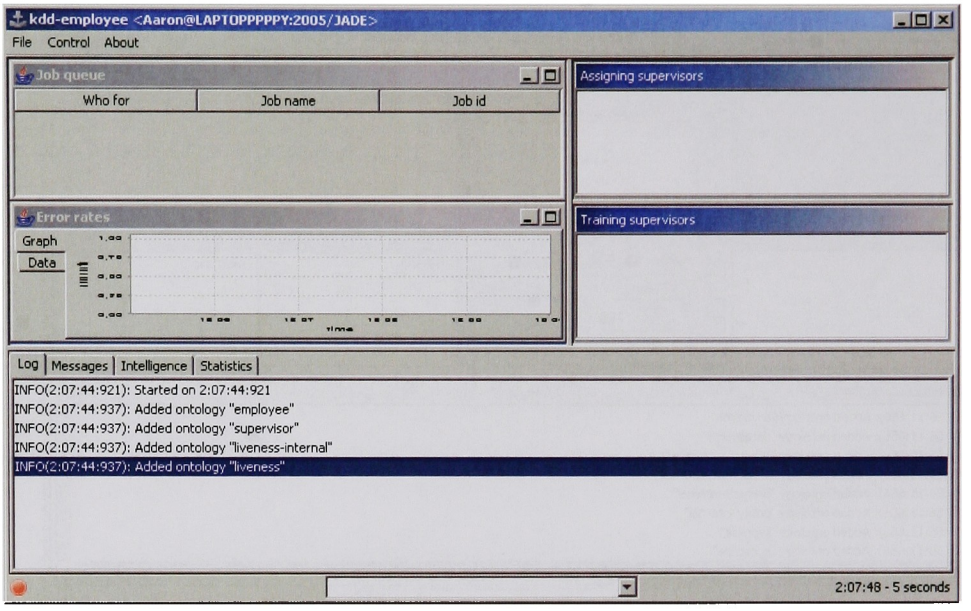


Figure B.6: Employee inactive

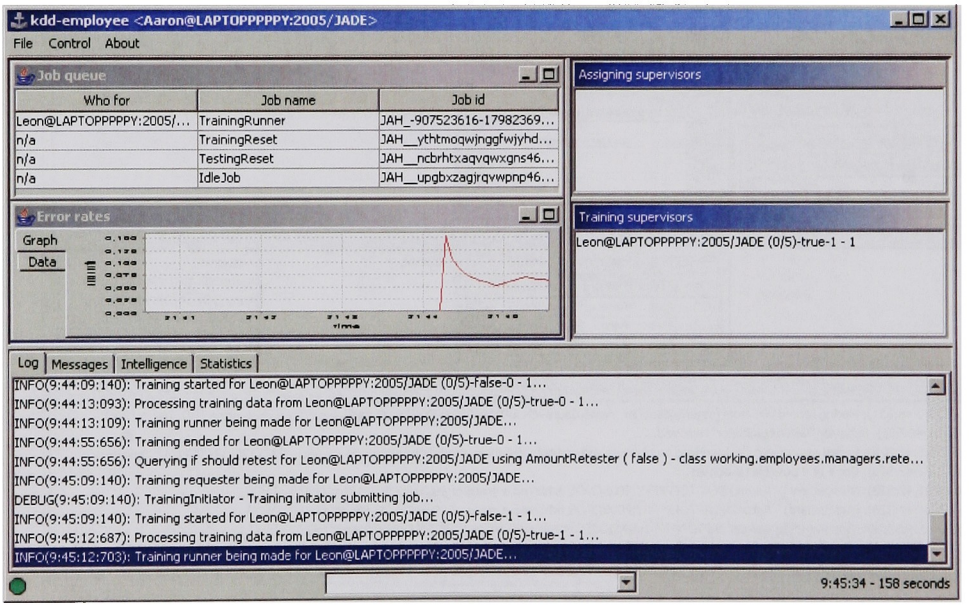


Figure B.7: Employee active

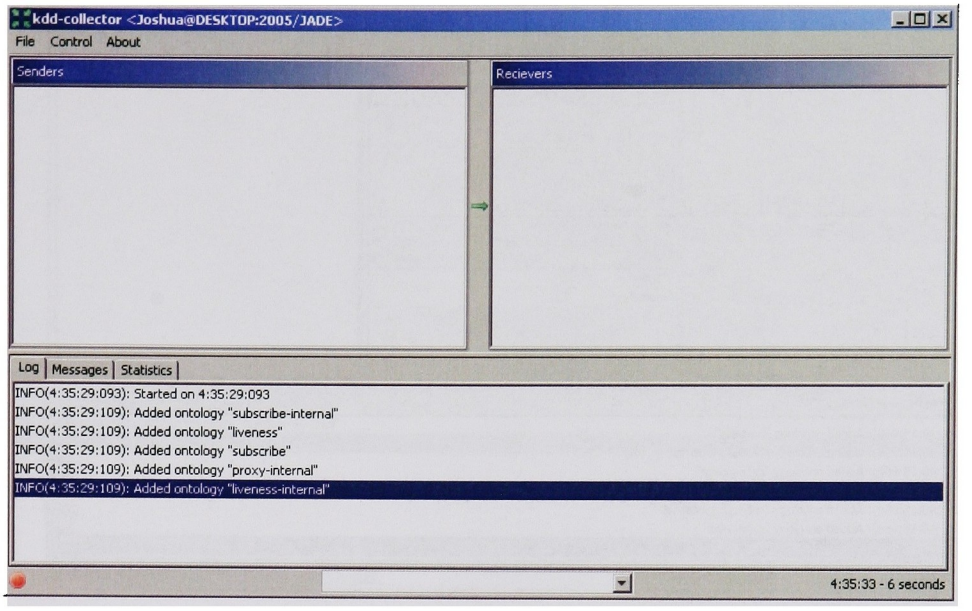


Figure B.8: Collector inactive

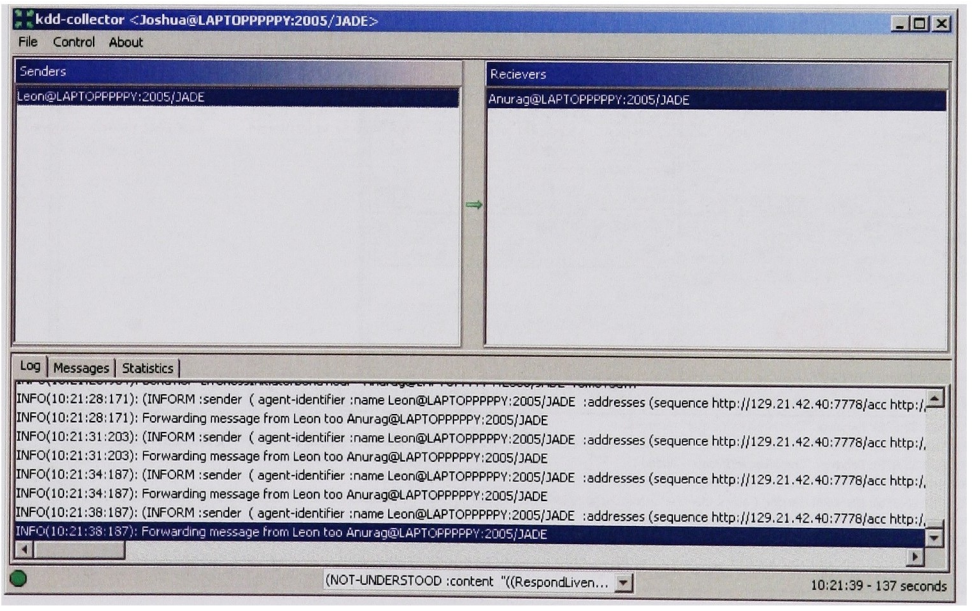


Figure B.9: Collector active

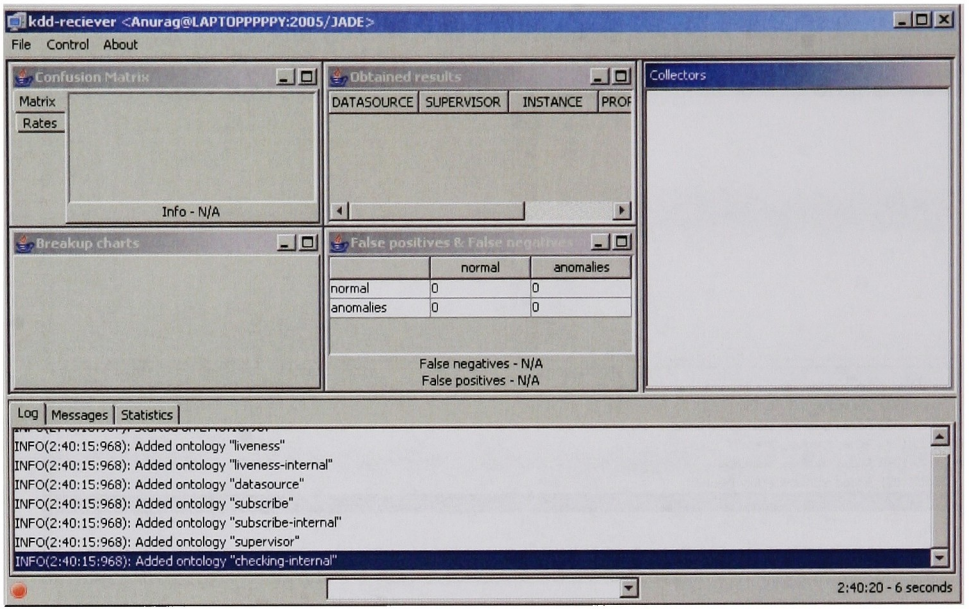


Figure B.10: Receiver inactive

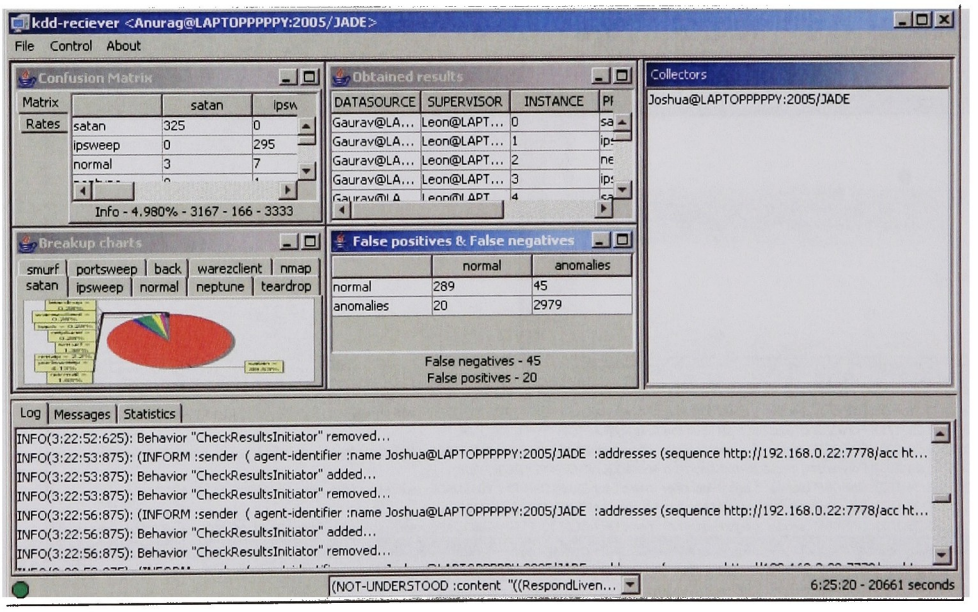


Figure B.11: Receiver active

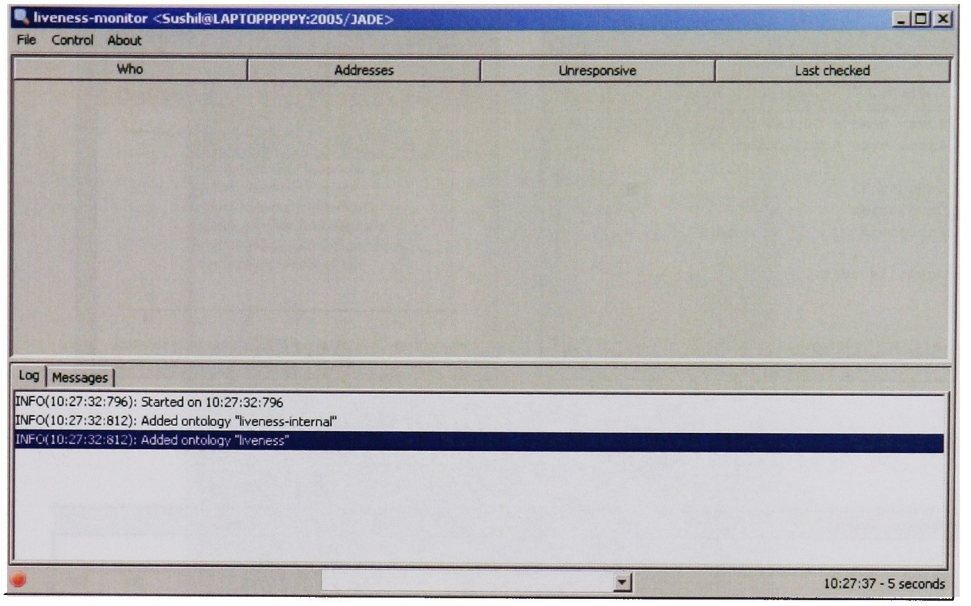


Figure B.12: Monitor inactive

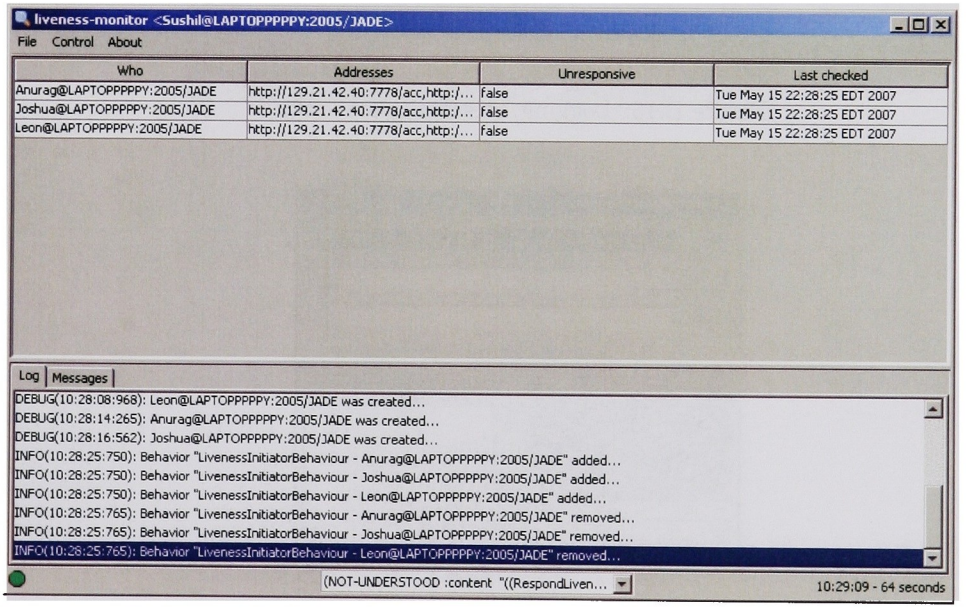


Figure B.13: Monitor active

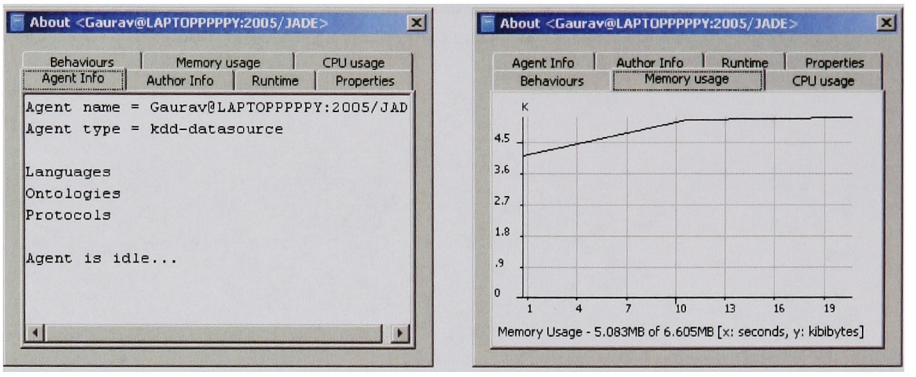


Figure B.14: Agent about menu

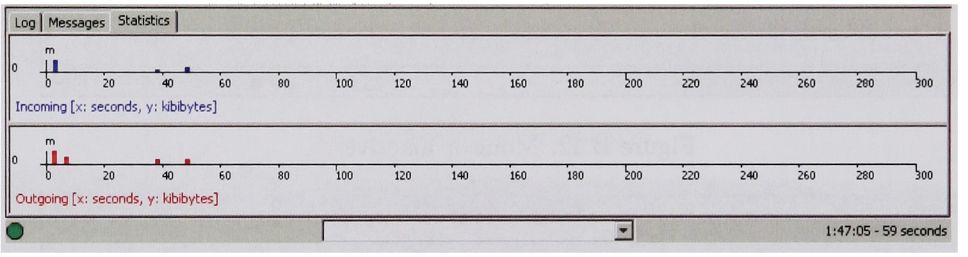


Figure B.15: Logging/messaging/statistics

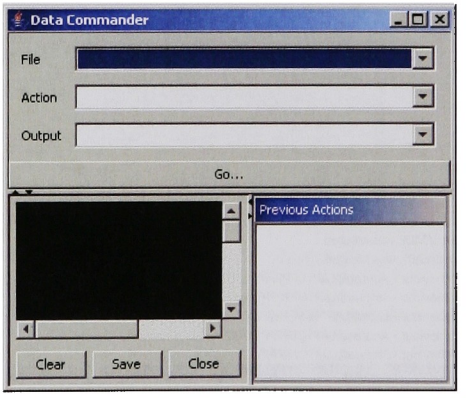


Figure B.16: Data processor

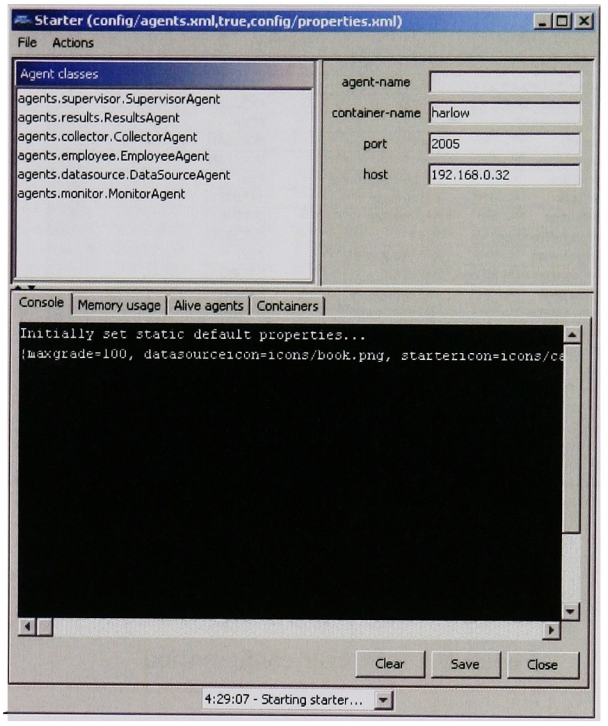


Figure B.17: Starter

Figure B.18: Starter main container

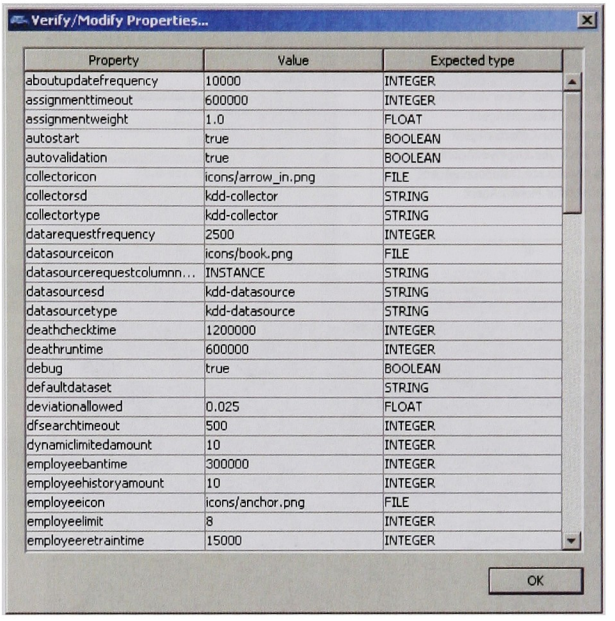


Figure B.19: Starter configuration

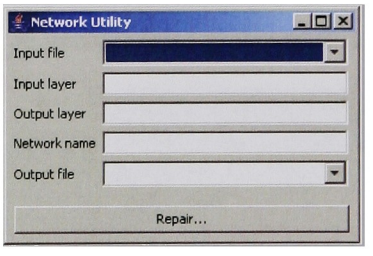


Figure B.20: Neural network repair

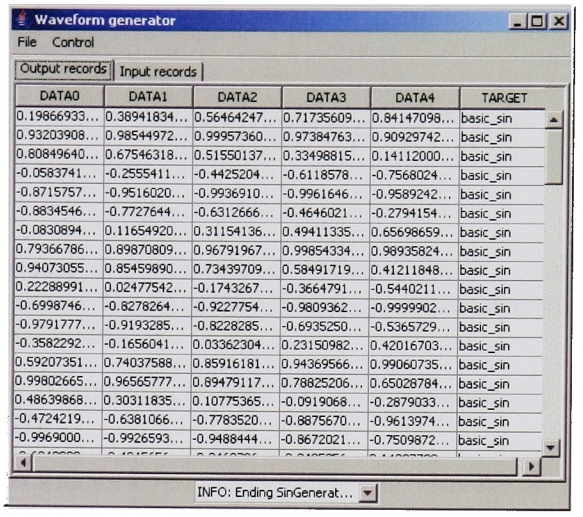


Figure B.21: Waveform generator

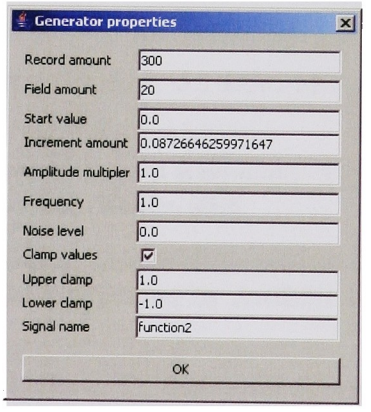


Figure B.22: Waveform generator configuration

This page intentionally left blank.

Bibliography

[1] BALASUBRAMANIYAN, J. S., GARCIA-FERNANDEZ, J. O., ISACOFF, D., SPAFFORD, E. H., AND ZAMBONI, D. An architecture for intrusion detection using autonomous agents. In *ACSAC* (1998), pp. 13–24.

[2] BELLIFEMINE, F., CAIRE, G., POGGI, A., AND RIMASSA, G. Jade: A white paper. *TILAB 3*, 3 (2003).

[3] BREESE, J., HECKERMAN, D., AND KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)* (San Francisco, CA, 1998), Morgan Kaufmann, pp. 43–52.

[4] BREIMAN, L., FRIEDMAN, J., STONE, C., AND OLSHEN, R. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[5] BROOKS, R. A. A robust layered control system for a mobile robot. Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.

[6] COLE, B., ECKSTEIN, R., ELLIOTT, J., LOY, M., AND WOOD, D. *Java Swing.* O'Reilly, 2002.

[7] COULOURIS, G., DOLLIMORE, J., AND KINDBERG, T. *Distributed Systems: Concepts and Design*, fourth ed. Addison Wesley, 2005.

[8] DEAN, T. Hierarchical bayesian models of the primate visual cortex. Keynote address at the Ninth International Symposium on Artificial Intelligence and Mathematics, January 2006.

[9] ENGELMORE, R., AND MORGAN, T. *Blackboard Systems.* Addison-Wesley, 1988.

[10] FABIO LUIGI BELLIFEMINE, GIOVANNI CAIRE, D. G. *Developing Multi-Agent Systems with JADE.* Wiley, 2007.

[11] FIPA. *FIPA ACL Message Structure Specification*, December 2002.

[12] FIPA. *FIPA Agent Message Transport Service Specification*, December 2002.

[13] FIPA. *FIPA Query Interaction Protocol Specification*, December 2002.

[14] FIPA. *FIPA Request Interaction Protocol Specification*, December 2002.

[15] FIPA. *FIPA SL Content Language Specification*, December 2002.

[16] FIPA. *FIPA Subscribe Interaction Protocol Specification*, December 2002.

[17] FIPA. *FIPA Agent Management Specification*, March 2004.

[18] FRAWLEY, W. J., PIATETSKY-SHAPIRO, G., AND MATHEUS, C. J. Knowledge discovery in databases - an overview. *Ai Magazine 13* (1992), 57–70.

[19] FREY, P. W., AND SLATE, D. J. Letter recognition using holland-style adaptive classifiers. *Machine Learning 6*, 2 (1991), 161–182.

[20] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, first ed. Addison-Wesley, 1995.

[21] GARCIA-MOLINA, H., ULLMAN, J. D., AND WIDOM, J. D. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[22] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA, 1989.

[23] GOLDMAN, C. V., AND ROSENSCHEIN, J. S. Incremental and Mutual Adaptation in Multiagent Systems. Tech. rep., The Hebrew University, 1996.

[24] GOLDMAN, C. V., AND ROSENSCHEIN, J. S. Mutually supervised learning in multiagent systems. In *Adaptation and Learning in Multi–Agent Systems*. Springer-Verlag, Berlin, 1996, pp. 85–96.

[25] GOODHILL, G., AND CARREIRA-PERPINAN, M. Cortical Columns, April 2002.

[26] GORODETSKY, V., LIU, J., AND SKORMIN, V. A., Eds. *Autonomous Intelligent Systems: Agents and Data Mining, International Workshop, AIS-ADM 2005, St. Petersburg, Russia, June 6-8, 2005, Proceedings* (2005), vol. 3505 of *Lecture Notes in Computer Science*, Springer.

[27] GROSSO, W. *Java RMI*, first ed. O'Reilly, 2001.

[28] HAN, J., AND KAMBER, M. *Data Mining: Concepts and Techniques*, first ed. Morgan Kaufmann, 2000.

[29] HAWKINS, J., AND BLAKESLEE, S. *On Intelligence*. Owl Books, 2005.

[30] HEATON, J. *Introduction to Neural Networks with Java.* Heaton Research, Inc., 2005.

[31] HUFFMAN, K., VERNOY, M., AND VERNOY, J. *Psychology in Action*, fifth ed. John Wiley & Sons, Inc., 2000.

[32] JIM, K.-C., AND GILES, C. L. How communication can improve the performance of multi-agent systems. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents* (New York, NY, USA, 2001), ACM Press, pp. 584–591.

[33] KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. P. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285.

[34] LASATER, C. G. *Design Patterns.* Wordware Publishing, 2006.

[35] LEE, W., AND STOLFO, S. Data mining approaches for intrusion detection. *Proceedings of the 7th USENIX Security Symposium 1* (1998), 26–29.

[36] LEI, J. Z., AND GHORBANI, A. Network intrusion detection using an improved competitive learning neural network. In *CNSR '04: Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR'04)* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 190–197.

[37] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability 1* (1967), 281–297.

[38] MARRONE, P. *The Complete Guide: All you need to know about Joone*, January 2007.

[39] MEANS, W. S., AND HAROLD, E. R. *XML in a Nutshell*, second ed. O'Reilly, 2002.

[40] MEDICINENET.COM. *Webster's New World Medical Dictionary.* Wiley Publishing, January 2003.

[41] MENCZER, F. Complementing search engines with online web mining agents. *Decis. Support Syst. 35*, 2 (2003), 195–212.

[42] MOUNTCASTLE, V. The columnar organization of the neocortex. *Brain 120*, 4 (1997), 701–722.

[43] NOVIKOV, D., YAMPOLSKIY, R. V., AND REZNIK, L. Anomaly detection based intrusion detection. In *ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 420–425.

[44] NUNES, L., AND OLIVEIRA, E. On learning by exchanging advice, 2002.

[45] NUNES, L., AND OLIVEIRA, E. Exchanging advice and learning to trust, 2003.

[46] OAKS, S., AND WONG, H. *Java Threads.* O'Reilly, 2004.

[47] PAN, Z., LIAN, H., HU, G., AND NI, G. An integrated model of intrusion detection based on neural network and expert system. In *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 671–672.

[48] PAN, Z.-S., CHEN, S.-C., HU, G.-B., AND ZHANG, D.-Q. Hybrid neural network and c4.5 for misuse detection. In *Machine Learning and Cybernetics* (2003), vol. 4, pp. 2463–2467.

[49] QUINLAN, J. R. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[50] RAMACHANDRAN, G., AND HART, D. A p2p intrusion detection system based on mobile agents. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference* (New York, NY, USA, 2004), ACM Press, pp. 185–190.

[51] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications* (2001), ACM Press, pp. 161–172.

[52] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science 2218* (2001), 329–350.

[53] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall, 2003.

[54] SCHLIMMER, J. C., AND FISHER, D. H. A case study of incremental concept induction. In *AAAI* (1986), pp. 496–501.

[55] SHANNON, C. E. A mathematical theory of communication. *Bell Systems Technical Journal 27*, 3 (July 1948), 379–423. Continued 27(4):623-656, October 1948.

[56] SIGILLITO, V., WING, S., HUTTON, L., AND BAKER, K. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest 10*, 3 (1989), 262–266.

[57] TAN, M. Multi-agent reinforcement learning: Independent vs. cooperative learning. In *Readings in Agents*, M. N. Huhns and M. P. Singh, Eds. Morgan Kaufmann, San Francisco, CA, USA, 1997, pp. 487–494.

[58] UTGOFF, P. E. Incremental induction of decision trees. *Machine Learning 4* (1989), 161–186.

[59] UTGOFF, P. E., BERKMAN, N. C., AND CLOUSE, J. A. Decision tree induction based on efficient tree restructuring. *Machine Learning 29*, 1 (1997), 5–44.

[60] VITAGLIONE, G., QUARTA, F., AND CORTESE, E. Scalability and performance of jade message transport system, 2002.

[61] WATKINS, C. J. C. H., AND DAYAN, P. Q-learning. *Machine Learning 8*, 3-4 (1992), 279–292.

[62] WEISS, G. Learning to coordinate actions in multi-agent systems. *IJCA193* (1993).

[63] WEISS, G. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.* MIT Press, 1999.

[64] WOOLDRIDGE, M., AND JENNINGS, N. R. Intelligent agents: Theory and practice. *Knowledge Engineering Review 10*, 2 (1995), 115–152.

[65] YOHANNES, Y., AND HODDINOTT, J. Classification and regression trees: An introduction. Tech. rep., International Food Policy Research Institute, 1999.